OPTIMIZATION OF DESIGN PARAMETERS OF COMPLEX HYDRAULIC SYSTEMS

By

TAIJOON UM

Bachelor of Science in Engineering
Seoul National University
Seoul, Korea
1977

Master of Science
Korea Institute of Science and Technology
Seoul, Korea
1979

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 1989

OPTIMIZATION OF DESIGN PARAMETERS OF

COMPLEX HYDRAULIC SYSTEMS

Thesis Approved:

Karl N. Reid

Thesis advisor

Lawrence J. Kolenosh

Gary L. Young

O. L. Fondry

Martin T Hagan

Norman N. Durham

Dean of the Graduate College

ACKNOWLEDGEMENTS

The author is very grateful to all who have encouraged him throughout his career. Their guidance and helping made possible this work.

Deep appreciation and gratitude is expressed to Dr. Karl N. Reid, thesis advisor and chairman of my advisory committee. His advice was invaluable and always encouraging during this research. I also would like to thank the rest of my committee members for their helpful comments and supports: Dr. Lawrence L. Hoberock, Dr. Richard L. Lowery, Dr. Gary E. Young, and Dr. Martin T. Hagen. In addition, I am grateful to Dr. Alim Kovalenko for his guidance during his short visit at Oklahoma State University. I also would like to thank my colleague, Mr. Kee-Hyun Shin for his helping in the various phases.

My family has been interested in my academic endeavors, and I thank them for their efforts. My wife, Hyesun, has sacrificed many years and I thank her for this love.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

A_i	functional representation of an algebraic equation
B_j	functional representation of a state equation
C_j	functional representation of a state equation
c_d	dimensionless drag coefficient for the motor
c_f	dimensionless friction coefficient for the motor
c_i	gradient vector
c_s	dimensionless slip coefficient for the motor
c_v	valve coefficient
c_0	isentropic speed of sound in the hydraulic fluid
D	differential operator ($\partial/\partial t$)
D_m	hydraulic motor displacement
e	error
F	functional representation of a constraint equation
f	functional representation of error
G	functional representation of a constraint equation
H	functional representation of combined equation
I	objective functional
J	motor and external load inertia
K	position feedback gain
K_1	valve flow gain coefficient
K_2	valve pressure flow sensitivity
K_p	pressure feedback gain
L	line length

l	number of parameters
l_s	number of parameters within the sth component
m	number of algebraic variables
m_s	number of algebraic variables within the sth component
N_m	hydraulic motor speed
n	number of state variables
n_s	number of state variables within the sth component
p	parameter
p_{ij}	jth parameter within the ith component
P	pressure
Q	volume flow rate
R_d	desired response
R_p	predicted response
r_s	number of independent variables within the sth component
r_0	inside radius of the line
s	number of components
t	independent variable
u	independent port variable
u_{ij}	jth independent port variable within the ith component
v	dependent port variable, volume of the fluid under compression
v_{ij}	jth dependent port variable within the ith component
x	state variable
x_{ij}	jth state variable within the ith component
Y_s	characteristic admittance

y	algebraic variable, algebraic objective function
y_{ij}	jth algebraic variable within the ith component
z	algebraic position
z_s	characteristic impedance
α	positive scalar
β	positive scalar
ε	small arbitrary quantity
γ	arbitrary function with continuous second derivative
λ	Lagrange multiplier
μ	viscosity of the hydraulic fluid
ν	kinematic viscosity of the hydraulic fluid
θ_m	position of the motor
ω_{c0}	Approximate Model Parameter
ψ	Lagrange multiplier
ζ_{c0}	Approximate Model Parameter

CHAPTER I

INTRODUCTION

The dynamic response of a dynamic system is governed by the design parameters of the components. The analytical design of a dynamic system is a two step process when the structure is given. First, the static performance requirements may be used to determine some design parameter values. Then, the dynamic response requirements must be considered.

If a desired response is specified, the necessary design parameters can be selected using an optimization method. Optimization might be accomplished by repeated simulation of the system equations using many possible sets of parameters. However, such a brute-force approach generally does not ensure satisfactory results and is not efficient if the system is fairly complex. In such cases, it is desirable to use a mathematical optimization technique. Existing optimization techniques based on variational approaches cannot be applied for the class of problems considered here because the algebraic variables are not explicit functions of the state variables.

Objective and Scope of Study

The objective of this study was the development of a numerical algorithm for the optimization of the design parameters of hydraulic systems which are modeled using a multiport concept. The modeling approach maintains the identity of the individual components. Although this study was motivated by a need for a new method for the optimal design of high performance hydraulic control systems, it was intended that the algorithm be applicable to other dynamic systems governed by dynamic models in the same class.

Some of parameters are not always available as continuous values. That is, components are manufactured in specific sizes. Therefore, the algorithm should also consider attributes of real, standard size, off-the-shelf components which have fixed parameters.

Major Results

A numerical algorithm was developed which can solve the parameter optimization problem for an hydraulic system which is modeled by coupled sets of state equations and algebraic equations. The variational approach combined with a conjugate gradient method forms the basic structure of the algorithm.

The error between the desired dynamic response and the predicted one is employed as an objective function having the form of a functional. The necessary conditions derived from the calculus of variations incorporated with an Implicit

Method have the form of a set of adjoint equations. An iterative algorithm is developed to solve these equations.

Application of the optimization algorithm is not restricted to hydraulic systems. It can be used as a design tool for other dynamic systems, such as pneumatic systems, electromechanical systems, etc, which are described by coupled sets of state equations and algebraic equations.

The optimization algorithm employs the concept of the conjugate gradient technique to improve the speed of convergence which would be slower with a simple gradient technique. Normally, the conjugate gradient technique is applicable to algebraic optimization problems only. However, in this study the use of a quadratic interpolation procedure allows the application of the conjugate gradient technique in dynamic system optimization.

The fixed parameters of actual components can be optimized as well as continuous parameters. The "branch and bound method" is modified to optimize the fixed parameters.

The optimization algorithm is demonstrated by means of two example hydraulic position control systems. The example systems are modeled by coupled sets of state and algebraic equations and have both continuous and fixed parameters.

CHAPTER II

PREVIOUS STUDIES

System optimization depends in part on system simulation. Simulation is necessary during an optimization process whether it is a one time process or repeated during every iteration.

Digital Simulation of Hydraulic Systems

A complex hydraulic system digital simulation program was developed by Smith (1) in 1975. He implemented an Implicit Method for the solution of the algebraic equations coupled with the state equations. In this method, algebraic equations are converted to state equations. Therefore, the method avoids iterative solutions of the algebraic equations and results in a reduction of computation time compared to the Newton-Raphson Method. Smith also developed a procedure for 'automatically' formulating the system equations based on a user-provided topological description of the system and an extensive component 'library'. Each component in the library was described by coupled sets of nonlinear algebraic and state equations using a multiport modeling approach.

In 1976, Ebbesen (2) extended the modeling approach used by Smith to include thermal effects. Though there are many

other studies concerning the simulation of hydraulic systems, the most relevant studies to this study are the ones discussed above.

Parameter Optimization in Dynamic Systems

The results of an extensive survey of literature relating to parameter optimization of hydraulic systems revealed that:

1. There have been many studies which deal with parameter optimization for the broad class of dynamic systems. Both algebraic and variational approaches have been used. All the studies identified, except one, use only state equations as constraints.
2. One study uses an algebraic optimization technique with both algebraic and state equations as constraints.

In 1976, Dransfield and Labrooy (3) studied the optimization of the design parameters in hydraulic control systems. An algebraic optimization technique known as COMPLEX was used to achieve the desired response. A time weighted error index served as a design criterion. COMPLEX is a self-learning procedure which rapidly converges on parameter values that minimize the chosen index. However, COMPLEX is an algebraic optimization method and requires the solution of (1+1)n equations before iteration procedure is started (Figure 1).

Several studies have been reported which dealt with parameter optimization of dynamic systems. In 1973, Ahmed and Georganas (4) derived the necessary conditions for optimal parameters from Gamkrelitze's generalized maximum principle (5). In 1974, Georganas (6) studied optimal parameter selection by an "imbedding technique". He used necessary conditions given by Boltyanskii (7) which allowed the solution for optimal parameters. However, there is no general way to obtain the analytical forms of these equations from the conditions. Therefore, the above two methods are restricted to specific problems and do not appear to be applicable to our class of optimization problems.

In 1976, Ahmed (8) proposed a simple gradient algorithm by applying the calculus of variations for the iterative solution of parameter optimization problems. The least squares estimation of system parameters was chosen as the cost functional. However, his algorithm lacked of generality in the sense that the constraint equations were only a set of differential equations. The simple gradient method used could be expected to cause relatively slow convergence.

In 1978, an analog computation scheme for optimization was suggested by Teo and Moore (9) utilizing the concept of directional derivatives. If the order of the system was n, the method required the forward-backward integration of a system of 2n differential equations to evaluate the direction of steepest descent of the objective functional. This method utilized the maximum principle where the Hamiltonian is

introduced. However, the constraints were only differential equations. Also, the optimization scheme was fairly complicated and the computation time was relatively long.

In 1980, Dolezal (10) presented a direct method for optimization by applying the calculus of variations to the same class of problems studied by Ahmed (8). He modified the direct scheme of Ahmed to obtain a solution for parameter optimization problems having a nonlinear cost functional and constraints. He also considered the parameter-dependent initial state of the systems. He showed that the method needed less computation time than that of Teo and Moore. Several examples were illustrated in detail to show the practical importance of his algorithm even though some error was found in the application. Dolezal's method required less initial formulation but the number of differential equations which require integration was larger than that of Teo and Moore. That is, if the order of the system was n and the number of parameters was l, the method required the forward integration of a system of (l+1)n differential equations to evaluate the direction of steepest descent. When the variational approach was used, the parameters only were perturbed. Had both the parameters and the state variables been perturbed, integration of only 2n equations would have been required.

In 1981, Orurk, Osipuv, and Petukhov (11) used a method of orthogonal projections for optimization of nonlinear control systems. The method can be used in conjunction with

nonlinear programming and it can handle random or regular searches. Differential equations were converted into finite difference equations and an algebraic optimization technique was applied. This method required the desired response to be input in the form of an equation of motion.

Another approach for parameter optimization of dynamic systems was presented by Gopalsami and Sanathanan (12) in 1985. Their performance criteria was similar to the ones used in the previous studies but the optimization technique used in their study was a search method which required repeated evaluation of performance within a specified range of the parameters. The constraint equations considered here were only linear differential equations.

A comparison chart of the algorithms discussed above is shown in Figure 1. The objective functions are almost identical in that they are of the form of a functional with respect to variables and parameters. Therefore, they are not included in the chart. The fourth column shows the constraint equations for each algorithm. Among the existing algorithms, only COMPLEX can handle the class of problems where the constraints consist of nonlinear algebraic equations as well as nonlinear differential equations. However, this algorithm did not use the variational approach and it is a relatively inefficient method to solve a functional optimization problem. It is seen in the fifth column that some algorithms require more computation time as the number of parameters is increased.

The present study has developed an algorithm based on a the variational approach to solve the class of problems of interest. The algorithm incorporates the Implicit Method developed by Smith to convert the algebraic equations to state equations. A conjugate gradient search technique is used to ensure rapid convergence.

Algorithm	Approach employed	Method based on	Constraint Equations		Number of Equations to be solved at each iteration
			Algebraic	State	
Dransfield & Labrooy	Algebraic	COMPLEX	✓	✓	2n
Ahmed & Georganas	Calculus of Variation	Explicit Solution		✓	-
Georganas	Calculus of Variation	Explicit Solution		✓	-
Ahmed	Calculus of Variation	Simple Gradient		✓	(l +1)n
Teo & Moore	Calculus of Variation	Projected Gradient		✓	2n+1
Dolezal	Calculus of Variation	Simple Gradient		✓	(l +1)n
Orurk,Osipuv & Petukhov	Algebraic	Projected Gradient		✓	l +n
Gopalsami & Sanathanan	Algebraic	Satisfactory Solution		✓	2 l n
Present Study	Calculus of Variation	Conjugate Gradient	✓	✓	2n

Figure 1. Comparison of Several Algorithms

n = number of system equations
l = number of parameters

CHAPTER III

ALGORITHM FOR PARAMETER OPTIMIZATION

The purpose of this study was to develop an algorithm which can select the design parameters by minimizing the chosen objective function. The objective function is based on the integration of the absolute error between the desired dynamic response and the predicted response of a hydraulic system. This problem is usually denoted as parameter optimization or optimal parameter estimation. The study included the development of a numerical iterative algorithm to optimize the design parameters of complex hydraulic systems and application of the algorithm to two examples to demonstrate its potential.

Problem Statement

Most high performance hydraulic control systems can be modeled by two sets of equations, i.e., differential and algebraic equations. A functional representation for these equations is

$$\dot{x}_i = A_i(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m, p_1, p_2, \ldots, p_l), \quad i=1,2,\ldots,n \quad (3.1)$$

$$0 = B_j(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m, p_1, p_2, \ldots, p_l), \quad j=1,2,\ldots,m \quad (3.2)$$

where x is a state variable, y is an algebraic variable, and p is a parameter. x, y, and p have dimensions n, m, and l, respectively. The objective functional has the form

$$I(p) = \int_0^t f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m, p_1, p_2, \ldots, p_l)\, dt \quad . \qquad (3.3)$$

The above integral is a function of x_i and y_j which are functions of p_k (k=1,2,...,l). Therefore, I(p) is a functional. The integrand of Equation (3.3) is an error, or the difference between the desired response and the predicted one.

The goal in optimization is to find the particular p which minimizes I(p), the objective functional. Minimizing I(p) means that the predicted response is close to the desired one or the desired response is obtained.

Error Index

Error is a mathematical expression defined by the difference between the desired and predicted response at a certain instant of time. Over the time period considered, the errors are simply summed.

There are several possible error indices for dynamic system optimization (Appendix A). The error index for this study has been chosen to be

$$I = \int_0^t |e|\, dt$$

$$= \int_0^t \left| R_d(x,y,p) - R_p(t) \right| dt \tag{3.4}$$

where e is the discrepancy between the desired response, R_d, and the predicted response, R_p.

The desired response may be given either in the form of an equation which is a function of time, or in the form of data with uncertainty which has discrete values at time intervals corresponding to those involved in the integration of the system equations.

Variational Approach

The system considered is assumed to have continuous variables x, y, and parameters p[1], where x and y are functions of p.

If the objective function is an algebraic function, then the necessary conditions may be derived by using classical calculus. That is, first derivatives with respect to each parameter are zero at minimum points. However, since the objective function is a functional, the variational approach is required. The fundamental idea of the calculus of variations, which deals with this class of problems, is to perturb the unknown variables to observe how the objective function behaves. By perturbing the variables, derivative-

---

[1]Some design parameters designate hydraulic component sizes (e.g., displacement of an hydraulic motor). Since components are available "off-the-shelf" in discrete or standard sizes, such parameters would not actually be continuous.

like conditions for the minimum can be derived for each variable.

Perturbed variables $\bar{x}_i$, $\bar{y}_j$ and $\bar{p}_k$ can be defined as

$$\bar{x}_i = \bar{x}_i (\bar{p}_k), \qquad i=1,2,\ldots,n \qquad (3.5)$$

$$\bar{y}_j = \bar{y}_j (\bar{p}_k), \qquad j=1,2,\ldots,m \qquad (3.6)$$

$$\bar{p}_k = p_k + \varepsilon_k^p \gamma_k^p, \qquad k=1,2,\ldots,l \qquad (3.7)$$

where p_k is the particular function that minimizes the integral I and satisfies the initial conditions, and γ_k^p is an arbitrary function with continuous second derivatives and some boundary conditions. The value ε_k^p represents a small arbitrary scalar. Therefore, Equations (3.5) through (3.7) represent arbitrary variables whose values are close to the optimum, which is illustrated in Figure 2.

A perturbation in p_k results in perturbations in x_i and y_j as follows,

$$\bar{x}_i = x_i + \varepsilon_i^x \gamma_i^x, \qquad i=1,2,\ldots,n \qquad (3.8)$$

$$\bar{y}_j = y_j + \varepsilon_j^y \gamma_j^y, \qquad j=1,2,\ldots,m \qquad (3.9)$$

where ε_i^x and ε_j^y represent small arbitrary scalars related to x_i, y_j, respectively, and also γ_i^x and γ_j^y represent arbitrary functions related to x_i, y_j, respectively.

Equation (3.2) can be written as

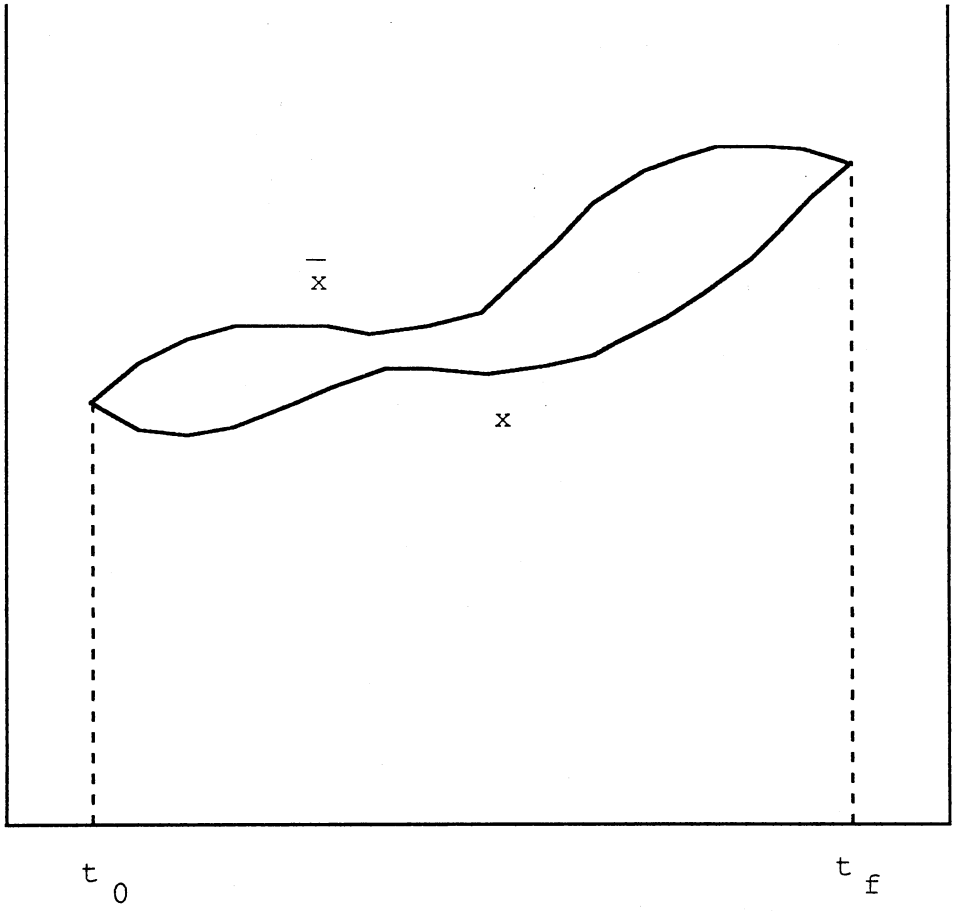


Figure 2. Arbitrary Function Close to Optimal Function

$$\dot{y}_j = C_j(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m, p_1, p_2, \ldots, p_l), \quad j=1,2,\ldots,m \quad (3.10)$$

where y_j is shown as a state variable. Converting Equation (3.2) into Equation (3.10) utilizes the concept of the Implicit Method and is discussed in Appendix B. The reason for having the additional state variables will be stated later.

An augmented integral is formed

$$I = \int_0^t (f + \sum_{i=1}^{n} \lambda_i G_i + \sum_{j=1}^{m} \psi_j F_j) dt, \quad (3.11)$$

where λ_i and ψ_j are the Lagrange multipliers for the constraint equations G_i and F_j, respectively and they are redefined here.

$$G_i = \dot{x}_i - A_i(x_i, y_j, p_k), \quad i=1,2,\ldots,n \quad (3.12)$$

$$F_j = \dot{y}_j - C_j(x_i, y_j, p_k), \quad j=1,2,\ldots,m \quad (3.13)$$

Since the integrand of Equation (3.11) is dependent on the functions $\bar{x}_i$, $\bar{y}_j$, and $\bar{p}_k$, where they are defined by Equations (3.7), (3.8), and (3.9), then the necessary conditions for a minimum can be obtained by setting to zero the derivatives of the integral I with respect to each ε as follows,

$$\frac{\partial I}{\partial \varepsilon_i^x} = \int_0^t \left[\sum_{j=1}^{n} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial \varepsilon_i^x} + \sum_{j=1}^{m} \frac{\partial f}{\partial y_j} \frac{\partial y_j}{\partial \varepsilon_i^x} \right.$$

$$+\sum_{k=1}^{n}\lambda_k\left(\sum_{j=1}^{n}\frac{\partial G_k}{\partial x_j}\frac{\partial x_j}{\partial \varepsilon_i^x}+\sum_{j=1}^{n}\frac{\partial G_k}{\partial \dot{x}_j}\frac{\partial \dot{x}_j}{\partial \varepsilon_i^x}\right)$$

$$+\sum_{k=1}^{n}\psi_k\left(\sum_{j=1}^{n}\frac{\partial F_k}{\partial x_j}\frac{\partial x_j}{\partial \varepsilon_i^x}+\sum_{j=1}^{n}\frac{\partial F_k}{\partial \dot{x}_j}\frac{\partial \dot{x}_j}{\partial \varepsilon_i^x}\right)\Bigg]\,dt$$

$$=\int_0^t\Bigg[\frac{\partial f}{\partial x_i}\gamma_i^x+\sum_{k=1}^{n}\lambda_k\left(\frac{\partial G_k}{\partial x_i}\gamma_i^x+\frac{\partial G_k}{\partial \dot{x}_i}\dot{\gamma}_i^x\right)$$

$$+\sum_{k=1}^{m}\psi_k\left(\frac{\partial F_k}{\partial x_i}\gamma_i^x+\frac{\partial F_k}{\partial \dot{x}_i}\dot{\gamma}_i^x\right)\Bigg]\,dt=0\,,\qquad i=1,2,\ldots,n \quad (3.14)$$

$$\frac{\partial I}{\partial \varepsilon_j^y}=\int_0^t\Bigg[\frac{\partial f}{\partial y_j}\gamma_j^y+\sum_{k=1}^{n}\lambda_k\left(\frac{\partial G_k}{\partial y_j}\gamma_j^y+\frac{\partial G_k}{\partial \dot{y}_j}\dot{\gamma}_j^y\right)$$

$$+\sum_{k=1}^{m}\psi_k\left(\frac{\partial F_k}{\partial y_j}\gamma_j^y+\frac{\partial F_k}{\partial \dot{y}_j}\dot{\gamma}_j^y\right)\Bigg]\,dt=0\,,\qquad j=1,2,\ldots,m \quad (3.15)$$

$$\frac{\partial I}{\partial \varepsilon_k^p}=\int_0^t\left[\frac{\partial f}{\partial p_k}\gamma_k^p+\sum_{i=1}^{n}\lambda_i\left(\frac{\partial G_i}{\partial p_k}\gamma_k^p\right)+\sum_{i=1}^{m}\psi_i\left(\frac{\partial F_i}{\partial p_k}\gamma_k^p\right)\right]dt=0\,,\qquad k=1,2,\ldots,l \quad (3.16)$$

It is noticed that $\partial F_k/\partial \dot{x}_i$ in Equation (3.14) and $\partial G_k/\partial \dot{y}_j$ in Equation (3.15) are equal to zero. By also noting

$$\frac{\partial G_i}{\partial \dot{x}_i}=1$$

and

$$\frac{\partial F_i}{\partial \dot{y}_j}=1\,,$$

Equations (3.14), (3.15), and (3.16) can be reduced to

$$\frac{\partial I}{\partial \varepsilon_i^x} = \int_0^t \left[\left(\frac{\partial f}{\partial x_i} + \sum_{k=1}^{n} \lambda_k \frac{\partial G_k}{\partial x_i} + \sum_{k=1}^{m} \psi_k \frac{\partial F_k}{\partial x_i} \right) \gamma_i^x + \lambda_i \dot{\gamma}_i^x \right] dt = 0$$

$$i=1,2,\ldots,n \qquad (3.17)$$

$$\frac{\partial I}{\partial \varepsilon_j^y} = \int_0^t \left[\left(\frac{\partial f}{\partial y_j} + \sum_{k=1}^{n} \lambda_k \frac{\partial G_k}{\partial y_j} + \sum_{k=1}^{m} \psi_k \frac{\partial F_k}{\partial y_j} \right) \gamma_j^y + \psi_j \dot{\gamma}_j^y \right] dt = 0$$

$$j=1,2,\ldots,m \qquad (3.18)$$

$$\frac{\partial I}{\partial \varepsilon_k^p} = \int_0^t \left[\left(\frac{\partial f}{\partial p_k} + \sum_{i=1}^{n} \lambda_i \frac{\partial G_i}{\partial p_k} + \sum_{i=1}^{m} \psi_i \frac{\partial F_i}{\partial p_k} \right) \gamma_k^p \right] dt = 0$$

$$k=1,2,\ldots,l \qquad (3.19)$$

Integrating by parts the last term in each integral of Equation (3.17) and (3.18) yields

$$\frac{\partial I}{\partial \varepsilon_i^x} = \int_0^t \left(\frac{\partial f}{\partial x_i} + \sum_{k=1}^{n} \lambda_k \frac{\partial G_k}{\partial x_i} + \sum_{k=1}^{m} \psi_k \frac{\partial F_k}{\partial x_i} \right) \gamma_i^x \, dt + \lambda_i \gamma_i^x \Big|_0^t - \int_0^t \frac{d\lambda_i}{dt} \gamma_i^x \, dt = 0 \qquad (3.20)$$

$$\frac{\partial I}{\partial \varepsilon_j^y} = \int_0^t \left(\frac{\partial f}{\partial y_j} + \sum_{k=1}^{n} \lambda_k \frac{\partial G_k}{\partial y_j} + \sum_{k=1}^{m} \psi_k \frac{\partial F_k}{\partial y_j} \right) \gamma_j^y \, dt + \psi_j \gamma_j^y \Big|_0^t - \int_0^t \frac{d\psi_j}{dt} \gamma_j^y \, dt = 0 \qquad (3.21)$$

$$\frac{\partial I}{\partial \varepsilon_k^p} = \int_0^t \left(\frac{\partial f}{\partial p_k} + \sum_{i=1}^{n} \lambda_i \frac{\partial G_i}{\partial p_k} + \sum_{i=1}^{m} \psi_i \frac{\partial F_i}{\partial p_k} \right) \gamma_k^p \, dt$$

$$+ \psi_k \gamma^p_k \Big|_0^t - \int_0^t \frac{d\psi_k}{dt} \gamma^p_k \, dt = 0 \quad . \tag{3.22}$$

Since $\gamma^x(0) = \gamma^y(0) = \gamma^p(0) = 0$, the second term in each of the equations will vanish if

$$\lambda_i(t) = 0 \quad , \tag{3.23}$$

and

$$\psi_j(t) = 0 \quad , \tag{3.24}$$

which will be used as the boundary conditions.

The necessary conditions may be rewritten as

$$\frac{\partial f}{\partial x_i} + \sum_{k=1}^{n} \lambda_k \frac{\partial G_k}{\partial x_i} + \sum_{k=1}^{m} \psi_k \frac{\partial F_k}{\partial x_i} - \frac{d\lambda_i}{dt} = 0 \quad , \tag{3.25}$$

$$\frac{\partial f}{\partial y_j} + \sum_{k=1}^{n} \lambda_k \frac{\partial G_k}{\partial y_j} + \sum_{k=1}^{m} \psi_k \frac{\partial F_k}{\partial y_j} - \frac{d\psi_j}{dt} = 0 \quad , \tag{3.26}$$

$$\frac{\partial f}{\partial p_k} + \sum_{i=1}^{n} \lambda_i \frac{\partial G_i}{\partial p_k} + \sum_{i=1}^{m} \psi_i \frac{\partial F_i}{\partial p_k} = 0 \quad . \tag{3.27}$$

The above equations are similar to Euler-Lagrange equations. If a function combining several functions is introduced, the equations can be made simpler, i.e.,

$$H(x,y,p,\lambda,\psi) = -f + \sum_{i=1}^{n} \lambda_i A_i + \sum_{j=1}^{m} \psi_j C_j \quad , \tag{3.28}$$

where λ_i and ψ_j are again the Lagrange multipliers and A_i and C_j are defined in Equations (3.1) and (3.10) respectively.

Then, the following equations are equivalent to Equations (3.25), (3.26), and (3.27).

$$-\frac{d\lambda_i}{dt} = \frac{\partial H}{\partial x_i}, \tag{3.29}$$

$$-\frac{d\psi_j}{dt} = \frac{\partial H}{\partial y_j}, \tag{3.30}$$

$$0 = \frac{\partial H}{\partial p_k}. \tag{3.31}$$

Equations (3.29) and (3.30) can be solved by backward integration using the boundary conditions given by Equations (3.23) and (3.24). However, Equation (3.31) is impossible to solve for p_k unless λ_i and ψ_j in Equations (3.29) and (3.30) are determined at values of p_k which satisfies Equation (3.31). The use of the gradient, $\partial H/\partial p_k$, to obtain the optimum is discussed in the following section.

Numerical Algorithm

Gradient Method

The derivative given by Equation (3.31) describes the decrease of the objective functional when the parameters are changed in an iterative manner using their gradients. Although a steepest descent (simple gradient) method can be applied in the optimization, it is not efficient in spite of its relative simplicity.

There are several other algorithms known to be more efficient than the steepest descent method. Conjugate gradient is one technique worthy of consideration. However, the direct application of this method to the present study is not possible since the objective function cannot be obtained as an explicit function of the system variables. However, the idea behind this method may be adapted for the numerical algorithm.

In the method of conjugate gradients for algebraic optimization, an arbitrary initial guess for the parameters is made. ("Arbitrary values" are constrained within a certain boundary to satisfy the physical conditions). Then, a sequence of parameters is evaluated successively using the following equation:

$$p_{i+1} = p_i + \alpha_i c_i, \qquad i=0,1,2,... \quad (3.32)$$

where α_i are positive scalars which define the distance between p_i and p_{i+1} along the gradient vector c_i.

The vector c_i is chosen initially as a negative vector of the gradient of the objective function, ∇y, and is given by the following equation

$$c_{i+1} = -\nabla y_{i+1} + \beta_i c_i, \qquad (3.33)$$

where β_i are positive scalars which are to be determined (Appendix C). Combining Equations (3.32) and (3.33) and substituting into the objective function y allows computation

of α_i by locating the parameters where the objective function is minimized (23).

The conjugate method can be adapted for optimization of the functional, I, even though the numerical algorithm does not allow α_i to be expressed explicitly in the objective function. Now, a numerical approach for computing α_i is required.

When three points are given for an algebraic function, the minimum of the function can be determined by quadratic interpolation. Similarly, if three different values of α_i are chosen, three corresponding values of I(p) can be obtained by simulation ("simulation" here simply means the integration of the original system equations). This set of values of α_i and I(p) will yield the value of α_i which minimizes I(p). Initial values of α_i may be chosen which are not less than zero. Also, they may be adjusted according to the current change of the parameters.

The above approach requires more simulation at each iteration of the optimization process. However, as far as overall computation time (until convergence occurs) is concerned, this algorithm is more efficient than when a simple gradient algorithm is used.

The functional optimization algorithm developed here has the following steps:

(1) Choose a set of parameters within feasible boundaries as initial guess (i.e., preliminary design values).

(2) Evaluate x, y, and the objective function I by

integrating Equations (3.1), (3.10), and (3.11) respectively.

(3) Evaluate λ and ψ by integrating Equations (3.29) and (3.30).

(4) Compute the gradients $\partial H/\partial p_k$ using Equation (3.31).

(5) Choose three values for α and determine new parameter values using Equation (3.32).

(6) Evaluate x,y, and the objective function by integrating Equations (3.1), (3.10), and (3.11) respectively using the new parameter values.

(7) Determine α required to minimize the objective function through quadratic interpolation.

(8) Calculate the updated parameters using Equation (3.32).

(9) Check if the parameters are within the user supplied boundaries and project them onto the boundaries if necessary.

(10) Check if the parameters converge by observing either the gradients or the error.

(11) Repeat (2) through (10) if necessary.

The diagram shown in Figure 3 summarizes the continuous parameter optimization algorithm.

Fixed Parameter Consideration

The design parameters are assumed to converge using the above algorithm. If the parameters are not of practical values, they should be forced (projected) into boundaries

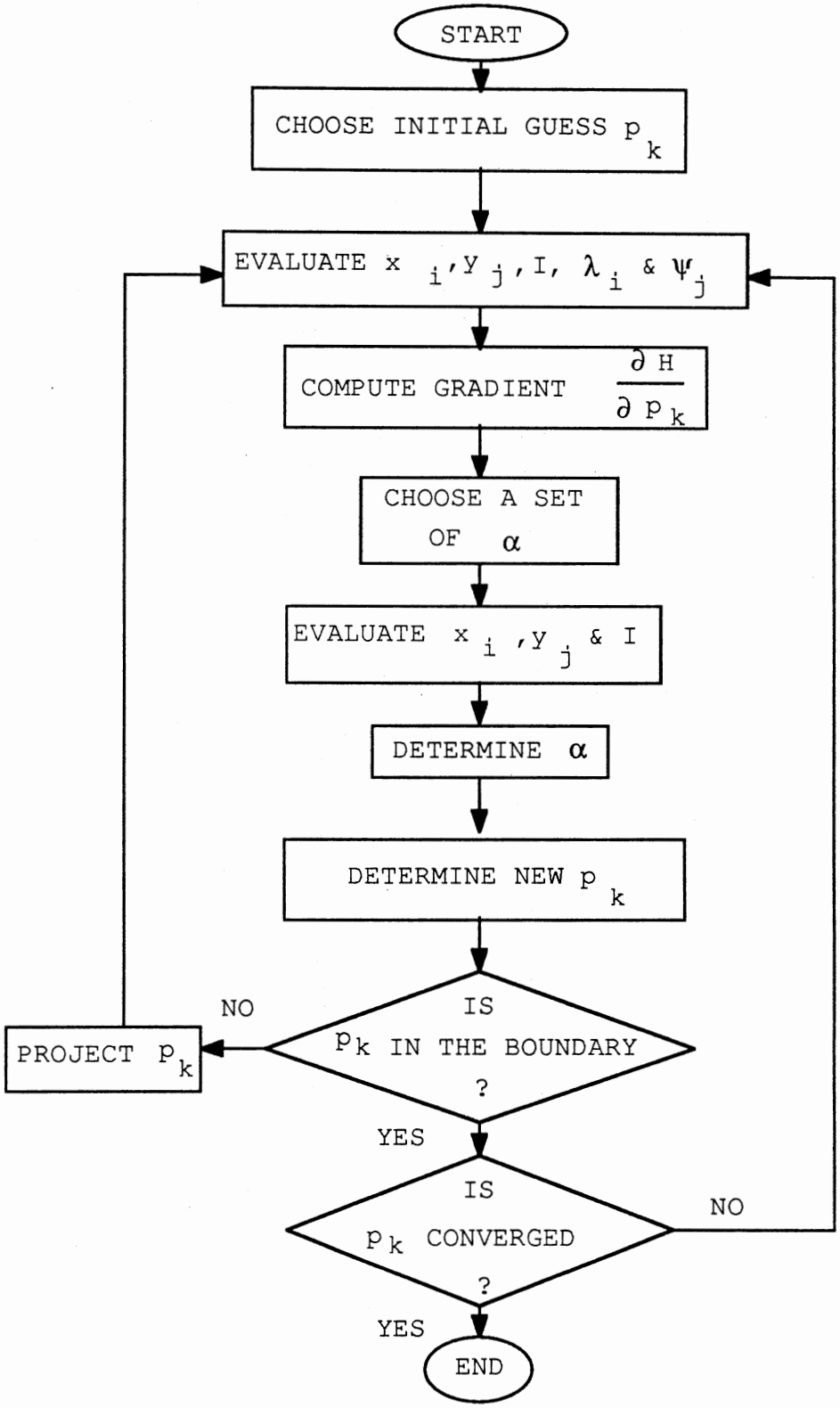


Figure 3. Algorithm for Parameter Optimization for Continuous Parameters

which define a practical range of values of the parameters. For this purpose, "hard" boundaries are easily imbedded in the algorithm. Once the boundary is established, a computed value outside the boundary is projected onto the boundary.

As mentioned in the previous section, actual design parameters, such as motor displacement, are not always available as continuous values. Normally motor displacements are only available in the form of standard off-the-shelf sizes. The simplest method is first to treat the parameters as continuous parameters, ignoring the discrete properties. Then the nearest feasible discrete values for the optimized parameters can be chosen. However, the nearest values may not be optimal.

Using the branch and bound method can ease the difficulties of such an optimization problem. The branch and bound method was developed by Land and Doig (33) to solve the linear integer programming problem. The method consists of first making a systematic search for continuous values of the parameters. If the continuous solution produces integer values, then they constitute the optimal solution. If not, then the two nearest integer values for each parameter are introduced to form two additional problems. The process of forming these subproblems is called "branching". The two integer values serve as upper and lower "bounds" respectively. This procedure of branch-and-bound is repeated until (a) branching violates the feasible boundaries, or (b) the smallest objective function is obtained.

The branch and bound algorithm is adapted for application with the available discrete values (not just integers) in this study. Its application is discussed in Chapter V.

CHAPTER IV

MODIFIED ALGORITHM FOR COMPONENT-ORIENTED MODEL

The purpose of this chapter is to show the modification of the optimization algorithm developed in Chapter III for a component-oriented model. An hydraulic system can be described by a component-oriented model.

Component-Oriented Model

Using a component-oriented model, the identity of each component can be maintained in the system equations. The variables associated with the kth component are the state variable x_{ki}, the algebraic variable y_{ki}, the parameter p_{ki}, the independent port variable u_{ki}, and the dependent port variable v_{ki}. The subscript ki indicates the ith variable in the kth component.

The equations which model the kth component are

$$\dot{x}_{ki} = A_{ki}(x, y, p, u, v), \tag{4.1}$$

$$0 = B_{kj}(x, y, p, u, v), \tag{4.2}$$

where kj indicates the jth variable in the kth component. Therefore, the redefined constraint equations have the forms of

$$G_{ki} = \dot{x}_{ki} - A_{ki}(x, y, p, u, v), \tag{4.3}$$

$$F_{kj} = \dot{y}_{kj} - C_{kj}(x, y, p, u, v) \quad , \tag{4.4}$$

y_{kj} can be determined by using the Implicit Method.

Application of the Implicit Method

The chain rule allows the derivative of Equation (4.2) to be written as

$$\frac{dB_{kj}}{dt} = 0 = \sum_{i=1}^{n_k} \frac{\partial B_{kj}}{\partial x_{ki}} \frac{dx_{ki}}{dt} + \sum_{i=1}^{m_k} \frac{\partial B_{kj}}{\partial y_{ki}} \frac{dy_{ki}}{dt} + \sum_{i=1}^{r_k} \frac{\partial B_{kj}}{\partial u_{ki}} \frac{du_{ki}}{dt} \quad . \tag{4.5}$$

It is noticed that the third term of the above equation is due to the existence of the independent variable u_{ki}. Equation (4.5) can then be rewritten as

$$\frac{dB_{kj}}{dt} = 0 = \sum_{i=1}^{n_k} \frac{\partial B_{kj}}{\partial x_{ki}} \frac{dx_{ki}}{dt} + \sum_{i=1}^{m_k} \frac{\partial B_{kj}}{\partial y_{ki}} \frac{dy_{ki}}{dt}$$

$$+ \sum_{i=1}^{r_k} \frac{\partial B_{kj}}{\partial u_{ki}} \left(\sum_{n=1}^{q_k} \frac{\partial u_{ki}}{\partial v_{kn}} \frac{dv_{kn}}{dt} \right) \quad . \tag{4.6}$$

Equation (4.7) can be formed from Equation (4.6) and used to solve for y_{ki}. The number of components is s, the number of x in the sth component is n_s, the number of y in the sth component is m_s, the number of p in the sth component is l_s, and the number of u in the sth component is r_s.

Evaluating λ, ψ (Equations (3.29) and (3.30)) requires the derivatives $\dot{y}$ with respect to the state variables, algebraic variables, and parameters. Since these derivatives

$$\begin{bmatrix} \frac{\partial B_{11}}{\partial y_{11}} & \frac{\partial B_{11}}{\partial y_{12}} \cdots \frac{\partial B_{11}}{\partial y_{1m_1}} & \frac{\partial B_{11}}{\partial y_{21}} & \frac{\partial B_{11}}{\partial y_{22}} \cdots & \frac{\partial B_{11}}{\partial y_{sm_s}} \\ \frac{\partial B_{12}}{\partial y_{11}} & \frac{\partial B_{12}}{\partial y_{12}} \cdots \frac{\partial B_{12}}{\partial y_{1m_1}} & \frac{\partial B_{12}}{\partial y_{21}} & \frac{\partial B_{12}}{\partial y_{22}} \cdots & \frac{\partial B_{12}}{\partial y_{sm_s}} \\ \vdots & & & & \vdots \\ \frac{\partial B_{1m_1}}{\partial y_{11}} & \frac{\partial B_{1m_1}}{\partial y_{12}} \cdots \frac{\partial B_{1m_1}}{\partial y_{1m_1}} & \frac{\partial B_{1m_1}}{\partial y_{21}} & \frac{\partial B_{1m_1}}{\partial y_{22}} \cdots & \frac{\partial B_{1m_1}}{\partial y_{sm_s}} \\ \frac{\partial B_{21}}{\partial y_{11}} & \frac{\partial B_{21}}{\partial y_{12}} \cdots \frac{\partial B_{21}}{\partial y_{1m_1}} & \frac{\partial B_{21}}{\partial y_{21}} & \frac{\partial B_{21}}{\partial y_{22}} \cdots & \frac{\partial B_{21}}{\partial y_{sm_s}} \\ \vdots & & & & \vdots \\ \frac{\partial B_{sm_s}}{\partial y_{11}} & \frac{\partial B_{sm_s}}{\partial y_{12}} \cdots \frac{\partial B_{sm_s}}{\partial y_{1m_1}} & \frac{\partial B_{sm_s}}{\partial y_{21}} & \frac{\partial B_{sm_s}}{\partial y_{22}} \cdots & \frac{\partial B_{sm_s}}{\partial y_{sm_s}} \end{bmatrix} \begin{bmatrix} \frac{dy_{11}}{dt} \\ \frac{dy_{12}}{dt} \\ \vdots \\ \frac{dy_{1m_1}}{dt} \\ \frac{dy_{21}}{dt} \\ \vdots \\ \frac{dy_{sm_s}}{dt} \end{bmatrix} =$$

$$\begin{bmatrix} -\sum_{i=1}^{n_s} \frac{\partial B_{11}}{\partial x_{1i}} \frac{dx_{1i}}{dt} - \sum_{i=1}^{r_s} \frac{\partial B_{11}}{\partial u_{1i}} \left(\sum_{j=1}^{q_s} \frac{\partial u_{1i}}{\partial v_{1j}} \frac{dv_{1j}}{dt} \right) \\ -\sum_{i=1}^{n_s} \frac{\partial B_{12}}{\partial x_{1i}} \frac{dx_{1i}}{dt} - \sum_{i=1}^{r_s} \frac{\partial B_{12}}{\partial u_{1i}} \left(\sum_{j=1}^{q_s} \frac{\partial u_{1i}}{\partial v_{1j}} \frac{dv_{1j}}{dt} \right) \\ \vdots \\ -\sum_{i=1}^{n_s} \frac{\partial B_{1m_1}}{\partial x_{1i}} \frac{dx_{1i}}{dt} - \sum_{i=1}^{r_s} \frac{\partial B_{1m_1}}{\partial u_{1i}} \left(\sum_{j=1}^{q_s} \frac{\partial u_{1i}}{\partial v_{1j}} \frac{dv_{1j}}{dt} \right) \\ -\sum_{i=1}^{n_s} \frac{\partial B_{21}}{\partial x_{2i}} \frac{dx_{2i}}{dt} - \sum_{i=1}^{r_s} \frac{\partial B_{21}}{\partial u_{2i}} \left(\sum_{j=1}^{q_s} \frac{\partial u_{2i}}{\partial v_{2j}} \frac{dv_{2j}}{dt} \right) \\ \vdots \\ -\sum_{i=1}^{n_s} \frac{\partial B_{sm_s}}{\partial x_{si}} \frac{dx_{si}}{dt} - \sum_{i=1}^{r_s} \frac{\partial B_{sm_s}}{\partial u_{si}} \left(\sum_{j=1}^{q_s} \frac{\partial u_{si}}{\partial v_{sj}} \frac{dv_{sj}}{dt} \right) \end{bmatrix} \qquad (4.7)$$

are not explicit functions of those variables, they should be derived using the Implicit Method again. Differentiating Equation (4.5) with respect to x, y, and p will form another equation that looks similar to Equation (4.7). Then, solving for $\partial\dot{y}/\partial x$, $\partial\dot{y}/\partial y$, and $\partial\dot{y}/\partial p$ and substituting them into Equation (3.31) will yield the gradients for the component-oriented model. The optimization steps in the previous chapter then can be applied.

A limitation of the optimization algorithm is that differential equations of a model should be transformed to state equations. Highly nonlinear differential equations may not be transformed to state equations. Also, nonlinear algebraic equations should be linearized if the variables of the nonlinear algebraic equations are linked with the differential equations in such a manner that nonlinearity of the algebraic equations prevents the differential equations from being transformed to state equations.

CHAPTER V

ILLUSTRATIVE EXAMPLES

The purpose of this chapter is to show the application of the optimization algorithm developed in this study to two important examples, each with two cases. The examples are typical electrohydraulic position control systems[1], and are within the class of problems of interest.

The basic system consists of an hydraulic power supply, an electrohydraulic servovalve controlled by an electronic servocontroller, and a rotary hydraulic motor driving an inertia load. Transmission lines can be installed between the servovalve and hydraulic motor. Position feedback is added to convert the open-loop 'rate type system' to a position control system. Pressure feedback can be added to enhance the degree of stability of the system. The system configuration is shown in Figure 4.

Example 1 - The System Without Transmission Lines

A multiport representation of the system is shown in Figure 5. Many studies have shown that the system dynamic

---

[1]The illustrative examples are based on the electro-hydraulic system installed on the east test stand in EN 215.

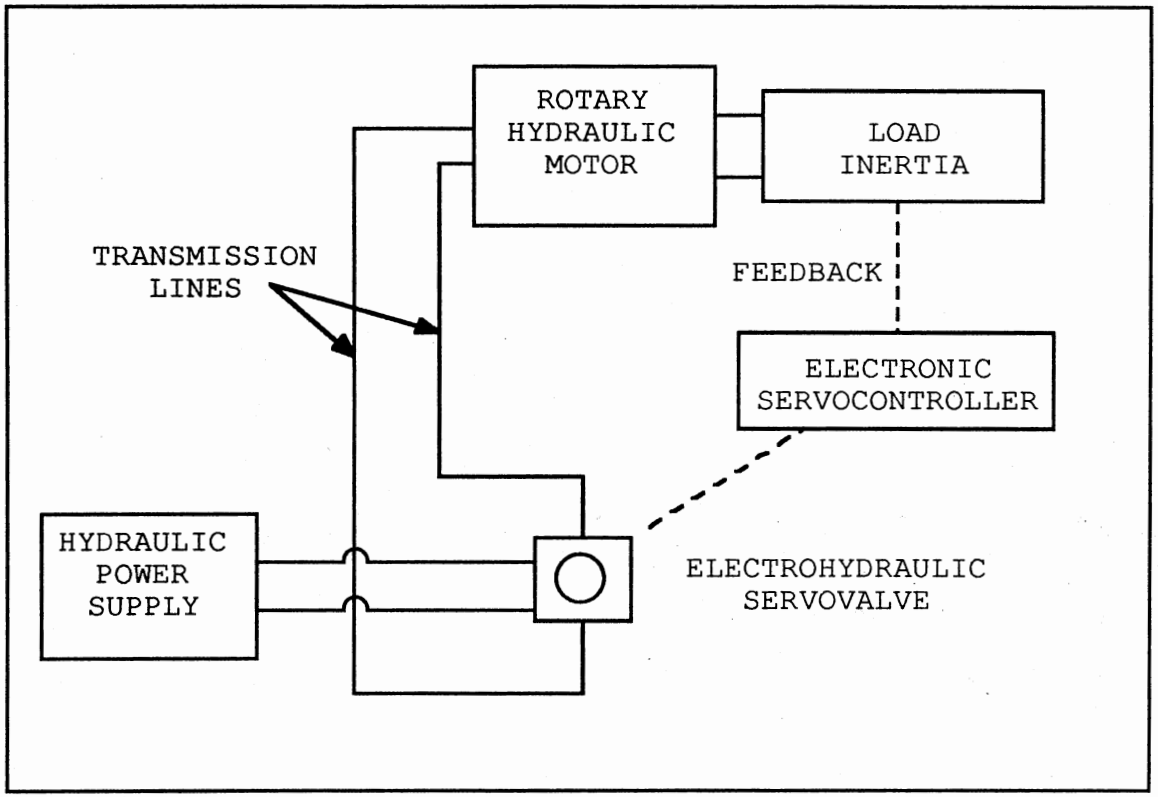


Figure 4. Electrohydraulic Position Control System

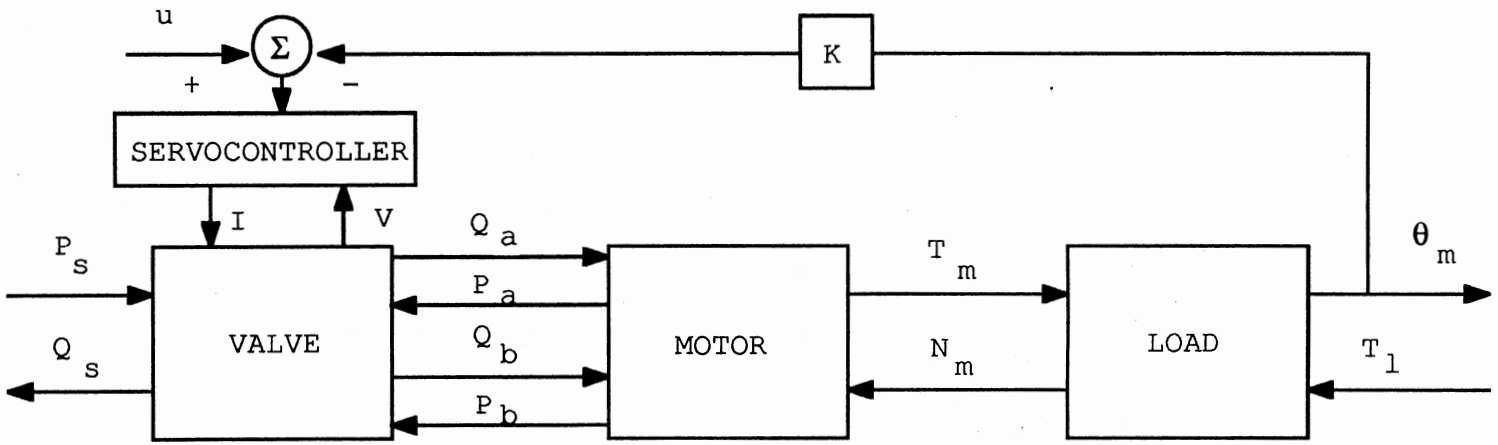


Figure 5. Multiport Representation of the System

behavior may be described for small excursions about an initial steady-state operating point by the following linearized equations:

Summing junction:

$$I = K_a(u - K\theta_m) \tag{5.1}$$

Valve characteristics:

$$Q_a = K_1 I + K_2 P_a \tag{5.2}$$

$$Q_b = K_1 I - K_2 P_b \tag{5.3}$$

Motor flow balance:

$$Q_a = \frac{v}{\beta}\frac{d}{dt}P_a + D_m N_m + c_s\frac{D_m(P_a - P_b)}{\mu} \tag{5.4}$$

$$Q_b = -\frac{v}{\beta}\frac{d}{dt}P_b + D_m N_m + c_s\frac{D_m(P_a - P_b)}{\mu} \tag{5.5}$$

Motor torque balance:

$$T_m = D_m(P_a - P_b) - c_d D_m \mu N_m - c_f D_m(P_a - P_b) \tag{5.6}$$

Load torque balance:

$$T_m = J\frac{dN_m}{dt} + T_l \tag{5.7}$$

Definition:

$$N_m = \frac{d\theta_m}{dt} \tag{5.8}$$

Definitions of all variables and parameters are given in the Nomenclature at the beginning of this dissertation.

Open-Loop System Dynamic Behavior

The open-loop behavior is explored first to form the basis for feedback control. A computer simulation for the nominal parameter values (those of the laboratory system) gives the open-loop velocity response to a step input in valve current I shown in Figure 6. The digital simulation program is listed in Appendix F. The following parameter values were used for the simulation:

I : change in input current (2 ma)

c_v : valve coefficient (0.216 $in^3/sec\text{-}ma\text{-}psi^{1/2}$)

v : volume of fluid under compression (25 in^3)

D_m : hydraulic motor displacement (0.01512 in^3/rad)

J : combined inertia of motor and external load (0.00173 $in\text{-}lbf\text{-}sec^2$)

K_a : amplifier gain (1.0)

β : bulk modulus of the fluid (165000 lbf/in^2)

μ : viscosity of fluid ($8x10^{-6}$ $lbf\text{-}sec/in^2$)

c_d : dimensionless drag coefficient for the motor ($1.12x10^5$)

c_f : dimensionless friction coefficient for the motor (0.17)

c_s : dimensionless slip coefficient for the motor ($4.4x10^{-9}$)

These values may be considered as the initial or preliminary design values for the parameters. It is assumed that the initial values of the parameters were chosen to meet steady-state performance requirements (not done here).

The open-loop dynamic response shown in Figure 6 indicates that the system is lightly damped, which is typical for electro-hydraulic systems of this class. Therefore, only

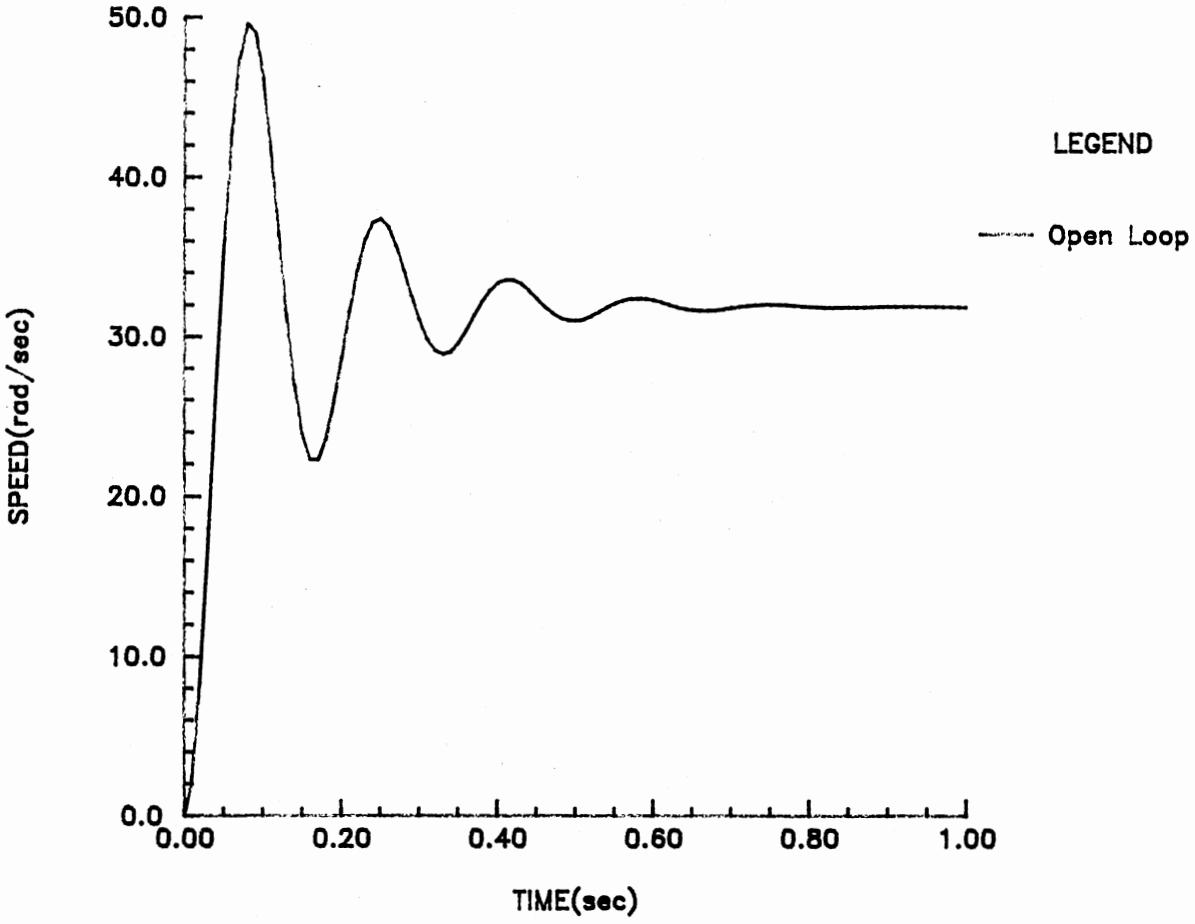


Figure 6. Response of the Open-Loop System to a Unit Step Input in Current to the Servovalve (With Initial Parameter Values)

limited position feedback may be used if the closed-loop system is to be stable.

Closed-Loop System Dynamic Behavior Using Initial Values of Parameters

Design of a closed-loop position control system involves a trade off between speed of response, degree of stability and load sensitivity. High loop gain normally is required to produce high speed of response and low steady-state load sensitivity (high stiffness). But, degree of stability normally is reduced with an increase in loop gain. The design problem is made more difficult if the open-loop system is lightly damped, as is the case in Example 1 and often in practice.

The result of a computer simulation with a position feedback gain of two and an amplifier gain of one is shown in Figure 7. All other parameter values are the same as for the open-loop system in the previous section. The digital simulation program is listed in Appendix F.

The closed-loop response is unstable for the loop gain chosen. It is assumed that the system loop gain was chosen to satisfy steady-state performance requirements, e.g., the minimum loop gain required to meet the requirement for steady-state load sensitivity.

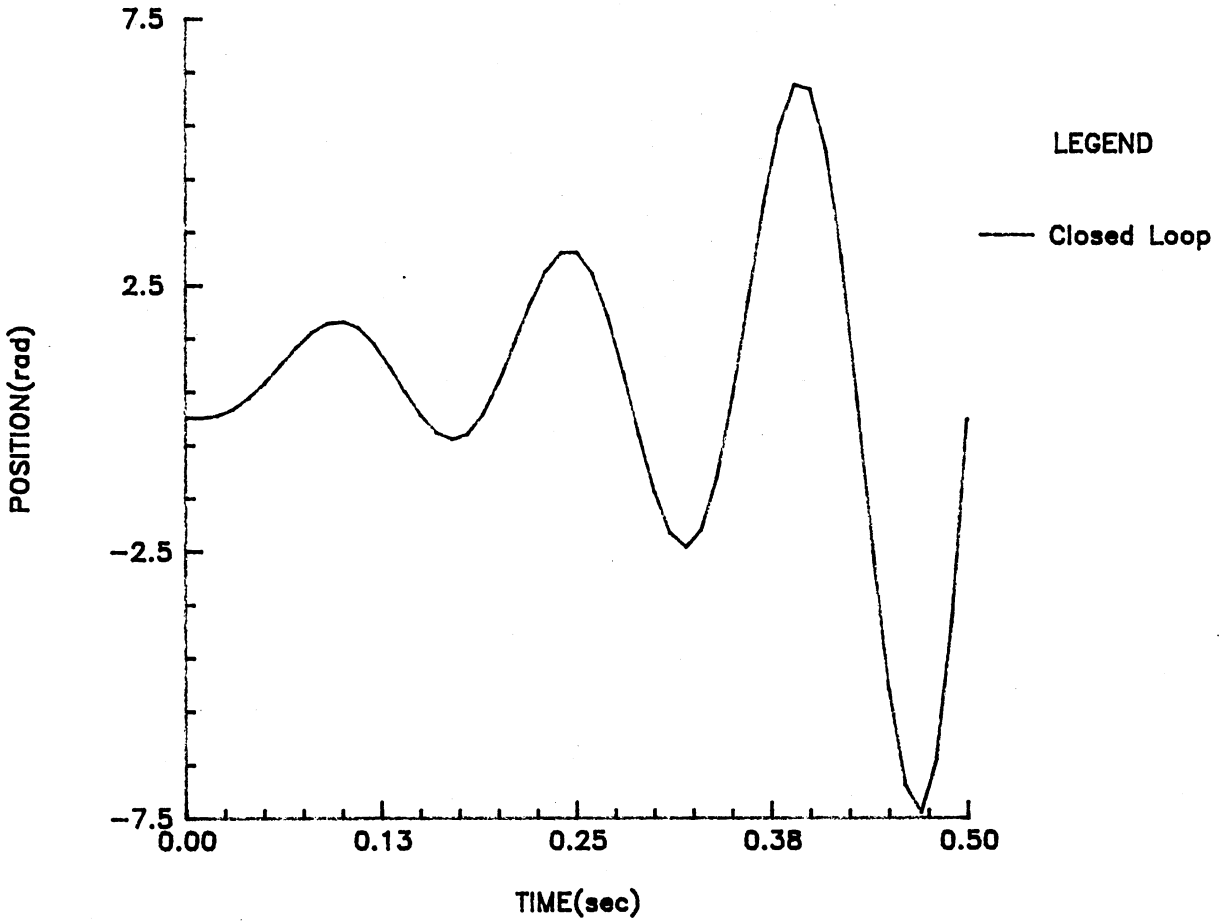


Figure 7. Response of the Position Feedback System to a Unit Step Input (Example 1; Initial Parameter Values; Without Pressure Feedback)

Basis for Dynamic Performance Improvement

The optimization algorithm developed in this study can be applied to obtain new parameter values which result in stable dynamic performance without sacrificing load sensitivity. Application of the algorithm requires the specification of a desired response, given either in equation form or tabulated values. The desired response is assumed to be given by the following equation:

$$R(t)= -2.2943 \exp(-36t) \sin(17.43t + 0.4510) +1. \quad (5.9)$$

The above equation describes the response of second order system with a steady-state position of 1 rad., a natural frequency of 40 rad/sec and a damping ratio of 0.9. A plot of R(t) is shown in Figure 8.

The desired response in Equation (5.9) demands relatively high performance from the system, i.e., both short rise time and high degree of stability. The purpose here is to demonstrate the benefits and limitations of the optimization algorithm. Clearly, a desired response can be chosen that is easily achievable. Likewise, a desired response can be chosen which is not achievable with any set of parameters within the available parameter space. In the latter case, the addition of some type of compensation or damping enhancement to the system will increase the likelihood of the dynamic performance being acceptable. But,

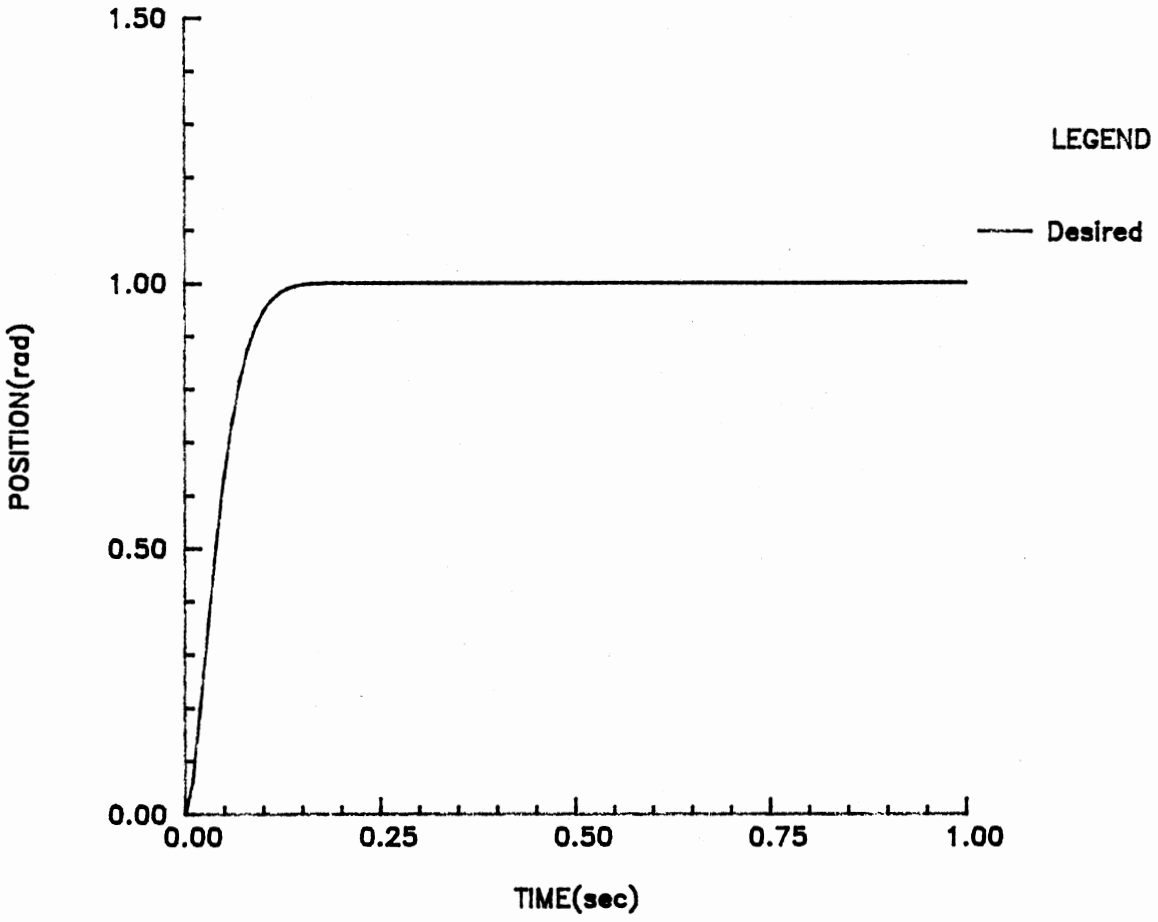


Figure 8. Desired Response of the Position Feedback System to a Unit Step Input

if an unrealistic desired response is chosen, there will not be a good solution.

Optimization of the Position Feedback System

Equations (5.1) through (5.8) must be rewritten as state equations for the optimization algorithm. For this purpose, the variables are redefined as follows,

$$x_1 = P_a - P_b$$

$$x_2 = N_m$$

$$x_3 = \theta_m$$

$$y_1 = Q_a$$

$$y_2 = Q_b .$$

For a system such as this, it can be demonstrated that a valid assumption is

$$Q_a = Q_b . \tag{5.10}$$

The state equations may be written as

$$\dot{x}_1 = 2\frac{\beta}{v}\left(y_1 - c_s \frac{D_m}{\mu} x_1 - D_m x_2 \right) \tag{5.11}$$

$$\dot{x}_2 = \frac{1}{J}\left\{ (1 - c_f) D_m x_1 - c_d \mu D_m x_2 \right\} \tag{5.12}$$

$$\dot{x}_3 = x_2 \tag{5.13}$$

$$y_1 = K_1 (u - K x_3) + \frac{1}{2} K_2 x_1 \tag{5.14}$$

The objective function can be written as

$$I = \int_0^t \left| x_3 - R(t) \right| dt \,, \tag{5.15}$$

where R(t) is the desired response given as Equation (5.9). From Equation (3.28), a combined function can be formed as

$$H = -\left| x_3 - R(t) \right| + \sum_{i=1}^{3} \lambda_i \dot{x}_i + \sum_{j=1}^{1} \psi_j \dot{y}_j \,, \tag{5.16}$$

where $\dot{x}_i$ are given in Equations (5.11) through (5.13).

The Implicit Method can be applied to transform Equation (5.14) to a state equation, with the result

$$\dot{y}_1 = -K K_1 \dot{x}_3 + \frac{1}{2} K_2 \dot{x}_1 \,, \tag{5.17}$$

or

$$\dot{y}_1 = -K K_1 x_2 + K_2 \frac{\beta}{v} \left(y_1 - c_s \frac{D_m}{\mu} x_1 - D_m x_2 \right) . \tag{5.18}$$

The state equations, Equations (5.11) through (5.13) and (5.18), can be substituted into the combined function, Equation (5.16). By differentiating the combined function according to Equations (3.29) and (3.30), the necessary conditions are obtained as follows,

$$\frac{d\lambda_1}{dt} = \lambda_1 \left(2 \frac{\beta}{v} c_s \frac{D_m}{\mu} \right) - \lambda_2 \left(\frac{1 - c_f}{J} D_m \right) + \psi_1 K_2 \frac{\beta}{v} c_s \frac{D_m}{\mu} \,, \tag{5.19}$$

$$\frac{d\lambda_2}{dt} = \lambda_1 \left(2 \frac{\beta}{v} D_m \right) + \lambda_2 \left(\frac{1}{J} c_d \mu D_m \right) - \lambda_3 + \psi_1 \left\{ K K_1 + K_2 \frac{\beta}{v} D_m \right\} \,, \tag{5.20}$$

$$\frac{d\lambda_3}{dt} = 1 \qquad \text{for } x_3 \geq R(t), \qquad \frac{d\lambda_3}{dt} = -1 \qquad \text{for } x_3 < R(t), \tag{5.21}$$

$$\frac{d\psi_1}{dt} = -\lambda_1\left(2\frac{\beta}{v}\right) - \psi_1 K_2 \frac{\beta}{v}. \tag{5.22}$$

Equations (5.19) through (5.22) are another set of differential equations for the Lagrange multipliers, λ and ψ, and are referred to as the adjoint equations.

For this example, it was assumed that all system parameters are fixed except as follows,

$p_1 = D_m$ (motor displacement),

$p_2 = K_1$ (valve flow gain coefficient).

Substituting the above definitions into the combined function (5.16) and differentiating the resulting equation with respect to each parameter, results in two gradient vectors as shown below:

$$\frac{\partial H}{\partial p_1} = -\lambda_1\left(2\frac{\beta}{v}\right)\left(\frac{c_s}{\mu}x_1 + x_2\right) + \lambda_2\frac{1}{J}\{(1-c_f)x_1 - c_d\mu x_2\}$$

$$+ \psi_1\frac{\beta}{v}\left\{-K_2\left(\frac{c_s}{\mu}x_1 + x_2\right)\right\}, \tag{5.23}$$

$$\frac{\partial H}{\partial p_2} = -\psi_1(Kx_2). \tag{5.24}$$

These gradients must be evaluated during each iteration in the optimization. The gradients approach zero, so that the last condition in Equation (3.31) is satisfied.

The boundaries for the parameters to be optimized were chosen arbitrarily as follows,

$$0.0095 \leq p_1 \geq 0.5 \quad \text{in}^3/\text{rad}$$

$$0.01 \leq p_2 \geq 1.0 \quad \text{in}^3/\text{ma}$$

The ranges could be chosen differently based on the judgement of the user and the available values for standard components. In actual practice, the parameter p_1 is available in discrete values only as shown in Table I. Therefore, the branch and bound method is employed to deal with this parameter.

TABLE I

TYPICAL AVAILABLE PARAMETER VALUES
(LARGER VALUES AVAILABLE ALSO)

Parameter	Available Values (in^3/rad)				
Motor Displacement, $p_1 = D_m$	0.009549	0.01512	0.030	0.05841	0.09549

A specific example serves to illustrate the branch and bound procedure to be used. Results of continuous optimization are examined to see if the discrete parameter p_1 falls between two available values. If not, p_1 is fixed to

the nearest available value and the parameter p_2 is optimized. If the p_1 falls between two available values, branching can be initiated. Two new subproblems are then generated. Suppose that continuous optimization results in the value $p_1 = 0.02$ in^3/rad. This value lies between the available values of 0.01512 and 0.03 in^3/rad (Table I); these available values are the nearest feasible bounds. An optimization problem with parameter p_1 fixed at 0.01512 is solved and another one with p_1 fixed at 0.03 is solved. For each problem, the optimization procedure produces an error. That parameter value which gives smallest error is the best choice.

The optimization procedure is summarized as follows:

(1) Choose a set of parameter values within the available boundaries (i.e., parameter space).

(2) Evaluate x_1, x_2, x_3, and y_1 by integrating Equations (5.11) through (5.13) and (5.18). Also evaluate the objective function given by Equation (5.15).

(3) Evaluate λ_1, λ_2, λ_3, and ψ_1 by integrating the adjoint Equations (5.19) through (5.22).

(4) Compute the gradients $\partial H/\partial p_i$ using Equations (5.23) and (5.24).

(5) Choose a set of three different values for α and calculate

$$p_{i+1} = p_i + \frac{\partial H}{\partial p_i} \alpha \tag{5.25}$$

(6) Repeat step (2) with the new parameter values from step (5).

(7) Determine the best value for α by minimizing the objective function, Equation (5.15), through quadratic interpolation.

(8) Calculate the updated parameters using Equation (5.25).

(9) Check if the parameters are within the allowable boundaries and project the parameters onto the boundaries if necessary.

(10) Check if the parameters converge by observing either the gradients or error. Go to (12) if parameters converge.

(11) Repeat step (2) through (10) until parameters converge.

(12) Apply the branch and bound method for discrete parameters. Divide into two new problems and repeat step (2) through (10) for each problem with the affected parameter fixed with the nearest available discrete values.

(13) Store the objective function value for each branch.

(14) Check if the branch is feasible or if branching produces optimum values. Stop if so.

(15) Repeat step (12) until the optimum discrete value is obtained.

A fourth order Runge-Kutta integration method was used to solve both the system equations and the adjoint equations. Definite integrals were evaluated using Simpson's rule.

Appendix H is a listing of the computer program used for the optimization.

Tabulated results of continuous optimization are shown in Table II. Application of the branch and bound method produced the new optimized parameters were obtained as follows,

$p_1 = 0.01512$ in^3/rad

$p_2 = 0.07118$ in^3/ma.

The dynamic response which results with the optimized continuous parameters is shown in Figure 9. In a least squares sense, the response matches the desired response fairly well, but the degree of stability is poor. The use of pressure feedback to produce a more acceptable response is illustrated in the following section.

Dynamic Performance Improvement through
Use of Pressure Feedback

Pressure feedback can be used to enhance the damping of the position control system. For this case, Equations (5.1), (5.14) and (5.18) must be rewritten as follows,

$$I = K_a\{u - K\theta_m - K_p(P_a - P_b)\}, \tag{5.1a}$$

$$y_1 = K_1(u - Kx_3 - K_p x_1) + \frac{1}{2}K_2 x_1,.$$

$$\dot{y}_1 = -K K_1 x_2 + (K_2 - 2K_1 K_p)\frac{\beta}{v}\left(y_1 - c_s\frac{D_m}{\mu}x_1 - D_m x_2\right).$$

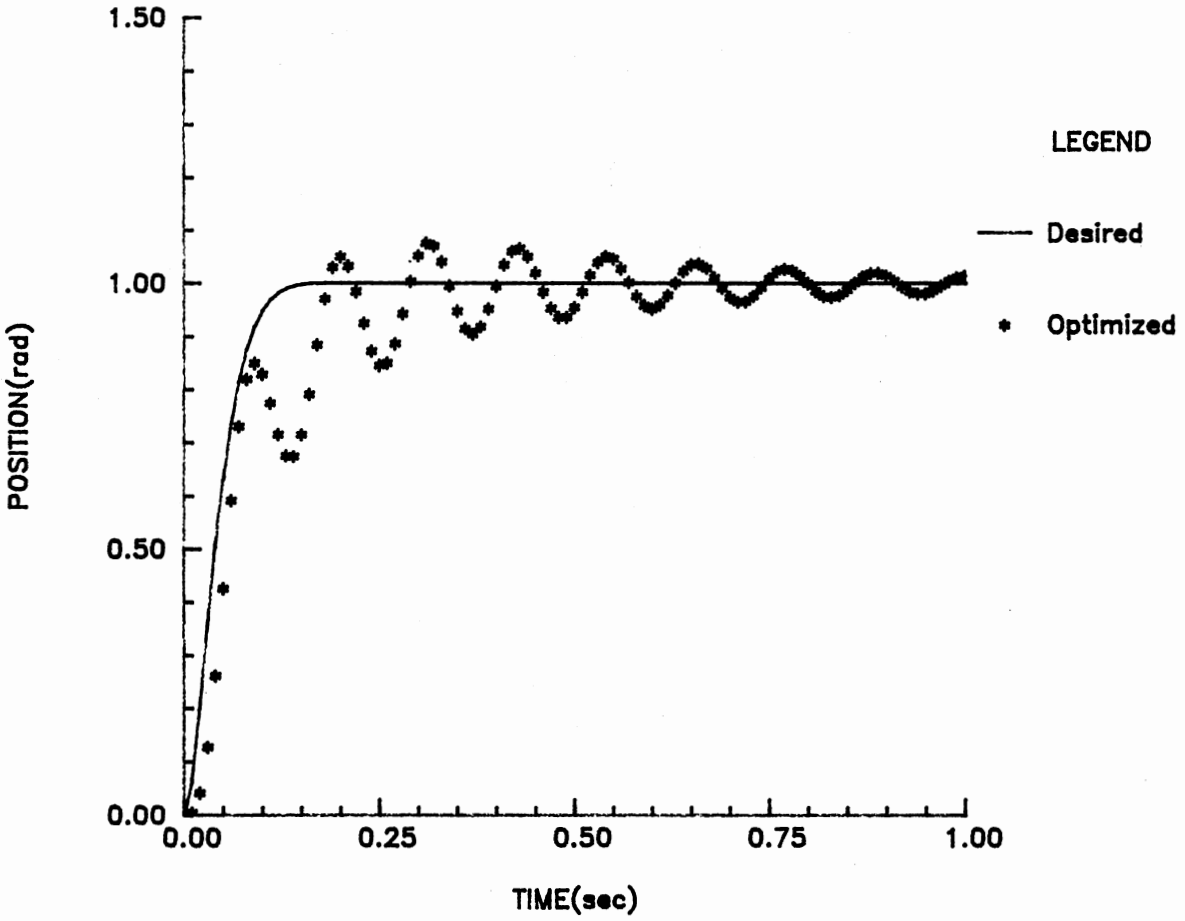


Figure 9. Response of the Position Feedback System to a Unit Step Input (Example 1; Optimized Continuous Parameter Values; Without Pressure Feedback)

The adjoint equations (5.19) through (5.22) are also rewritten as

$$\frac{d\lambda_1}{dt} = \lambda_1\left(2\frac{\beta}{v}c_s\frac{D_m}{\mu}\right) - \lambda_2\left(\frac{1-c_f}{J}D_m\right) + \psi_1\left(K_2 - 2K_1K_p\right)\frac{\beta}{v}c_s\frac{D_m}{\mu},$$

$$\frac{d\lambda_2}{dt} = \lambda_1\left(2\frac{\beta}{v}D_m\right) + \lambda_2\left(\frac{1}{J}c_d\mu D_m\right) - \lambda_3 + \psi_1\left\{KK_1 + \left(K_2 - 2K_1K_p\right)\frac{\beta}{v}D_m\right\},$$

$$\frac{d\lambda_3}{dt} = 1 \quad \text{for } x_3 \geq R(t), \qquad \frac{d\lambda_3}{dt} = -1 \quad \text{for } x_3 \geq R(t),$$

$$\frac{d\psi_1}{dt} = -\lambda_1\left(2\frac{\beta}{v}\right) - \psi_1\left(K_2 - 2K_1K_p\right)\frac{\beta}{v}.$$

Since pressure feedback is added to enhance the dynamic performance, the feedback gain K_p is a appropriate additional parameter to be optimized; here $K_p = p_3$.

New gradient vectors resulting from rewriting Equations (5.23) and (5.24), are

$$\frac{\partial H}{\partial p_1} = -\lambda_1\left(2\frac{\beta}{v}\right)\left(\frac{c_s}{\mu}x_1 + x_2\right) + \lambda_2\frac{1}{J}\left\{\left(1-c_f\right)x_1 - c_d\mu x_2\right\} - \psi_1\left(K_2 - 2K_1K_p\right)\frac{\beta}{v}\left(\frac{c_s}{\mu}x_1 + x_2\right),$$

$$\frac{\partial H}{\partial p_2} = -\psi_1\left\{Kx_2 + K_p 2\frac{\beta}{v}\left(y_1 - c_s\frac{D_m}{\mu}x_1 - D_m x_2\right)\right\}.$$

The new parameter requires the additional gradient vector:

$$\frac{\partial H}{\partial p_3} = -\psi_1\left\{K_1 2\frac{\beta}{v}\left(y_1 - c_s\frac{D_m}{\mu}x_1 - D_m x_2\right)\right\}.$$

The optimization procedure described in the previous section was applied to the basic feedback system augmented with pressure feedback. Tabulated results of the continuous optimization are shown in Table III. Appendix I is a listing of the computer program used for the optimization.

The response of the system to a step change in the system input in Figure 10 was determined by simulation using the continuous optimization results. Plots are shown for (1) the 'initial values' or 'preliminary design values' of the system parameters, and (2) the optimized parameters. Clearly, the preliminary design values result in an unacceptable dynamic performance. Alternatively, the optimized parameters produce an acceptable dynamic performance. A comparison of Figures 9 and 10 shows that the addition of pressure feedback results in a more acceptable degree of stability. (Not shown or considered here is that the use of pressure feedback results in a degradation in the system stiffness to load torque changes. A preferable approach would be to augment the basic system with dynamic pressure feedback or state feedback instead of pressure feedback. But, the purpose of the example is to illustrate the use of the optimization algorithm and not to suggest the best approach to achieving high performance.)

A limitation of most optimization algorithms, including the one developed herein, is that the so-called 'optimized parameters' may not be the best parameters within the user-prescribed parameter space. That is, the optimization can

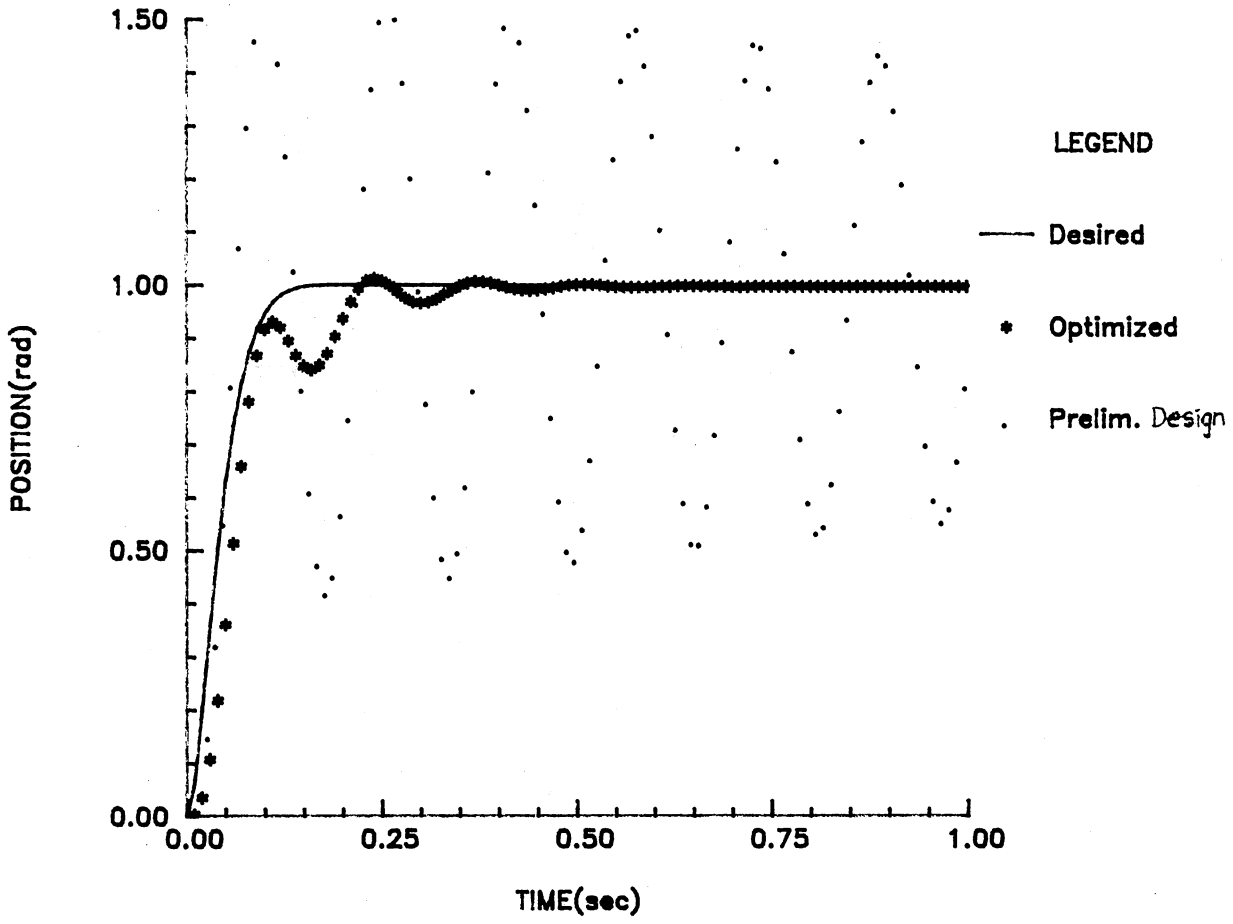


Figure 10. Response of the Position Feedback System to a Unit Step Input (Example 1; Optimized Continuous Parameter Values; With Pressure Feedback)

TABLE II

RESULTS OF CONTINUOUS OPTIMIZATION FOR EXAMPLE 1
WITHOUT PRESSURE FEEDBACK

iteration	p_1	p_2	error
1	0.0175	0.2	0.649
5	0.01977	0.1721	0.1465
9	0.02050	0.1623	0.0959
13	0.02092	0.1555	0.0762
17	0.02123	0.1501	0.0656
21	0.02148	0.1451	0.0598
25	0.02163	0.1407	0.0564
29	0.02177	0.1363	0.0542

TABLE III

RESULTS OF CONTINUOUS OPTIMIZATION FOR EXAMPLE 1
WITH PRESSURE FEEDBACK

iteration	p_1	p_2	p_3	error
1	0.015	0.25	0.005	0.2451
3	0.01666	0.2369	0.00549	0.1089
5	0.01747	0.2332	0.00560	0.0801
7	0.01806	0.2299	0.00582	0.0643
9	0.01847	0.1997	0.00593	0.0469
11	0.01875	0.1645	0.00599	0.0355
13	0.01886	0.1513	0.00600	0.0336
15	0.01887	0.1420	0.00610	0.0335

result in parameter values associated with a 'local minimum' instead of a 'global minimum'. If a number of local minima exist, and this is the norm in problems such as those treated here, the optimized parameter values will depend on the 'initial values' or what might be considered as the 'preliminary design values'.

The optimization algorithm was applied to Example 1 (case with pressure feedback). The results for the principal initial values are shown in Table III. Results for three additional sets of initial values are given in Appendix E. In all four cases convergence occurred, but to different sets of final values. One of these minima could be the global minimum within the specified parameter space, but there is no way of knowing from these results. It is possible that an efficient search procedure could be developed which find the global minimum within the parameter space. An investigation for such a procedure is included in the recommendations for further study. However, the local minima problem may not be too important since the experience and judgement of the user will always tend to be an important overlay on the optimization approach. Also, it is clear that there may in fact be several good solutions for any specific problem.

Example 2 - The System With Transmission Lines

This example is similar to the one considered in Example 1 except that appreciable length transmission lines are included between the servovalve and the motor. The system

multiport presentation is shown in Figure 11. Pressure feedback is not shown in Figure 11, but will be included in the second portion of the subsequent analysis

Experience has shown that the line dynamics can be described adequately using a zeroth order rational approximate model developed by Gerlach (34). Each connecting line is a two-port component, and is described by a pair of equations. The model is as follows,

$$P_a = \frac{Z_s T_e D}{1 + \frac{2\zeta_{c0}}{\omega_{c0}} D + \frac{1}{\omega_{c0}^2} D^2} Q_a + \frac{1}{1 + \frac{2\zeta_{c0}}{\omega_{c0}} D + \frac{1}{\omega_{c0}^2} D^2} P_c \tag{5.27}$$

$$Q_c = \frac{1}{1 + \frac{2\zeta_{c0}}{\omega_{c0}} D + \frac{1}{\omega_{c0}^2} D^2} Q_a + \frac{(-Y_s) T_e D}{1 + \frac{2\zeta_{c0}}{\omega_{c0}} D + \frac{1}{\omega_{c0}^2} D^2} P_c \tag{5.28}$$

$$Q_b = \frac{1}{1 + \frac{2\zeta_{c0}}{\omega_{c0}} D + \frac{1}{\omega_{c0}^2} D^2} Q_d + \frac{(-Y_s) T_e D}{1 + \frac{2\zeta_{c0}}{\omega_{c0}} D + \frac{1}{\omega_{c0}^2} D^2} P_b \tag{5.29}$$

$$P_d = \frac{Z_s T_e D}{1 + \frac{2\zeta_{c0}}{\omega_{c0}} D + \frac{1}{\omega_{c0}^2} D^2} Q_d + \frac{1}{1 + \frac{2\zeta_{c0}}{\omega_{c0}} D + \frac{1}{\omega_{c0}^2} D^2} P_b \tag{5.30}$$

where D is a differential operator, i.e., $\partial/\partial t$. ζ_{c0} and ω_{c0} are functions of a single dimensional group which is $\nu L/c_0 r_0^2$; this parameter is called the line damping number. Here ν is

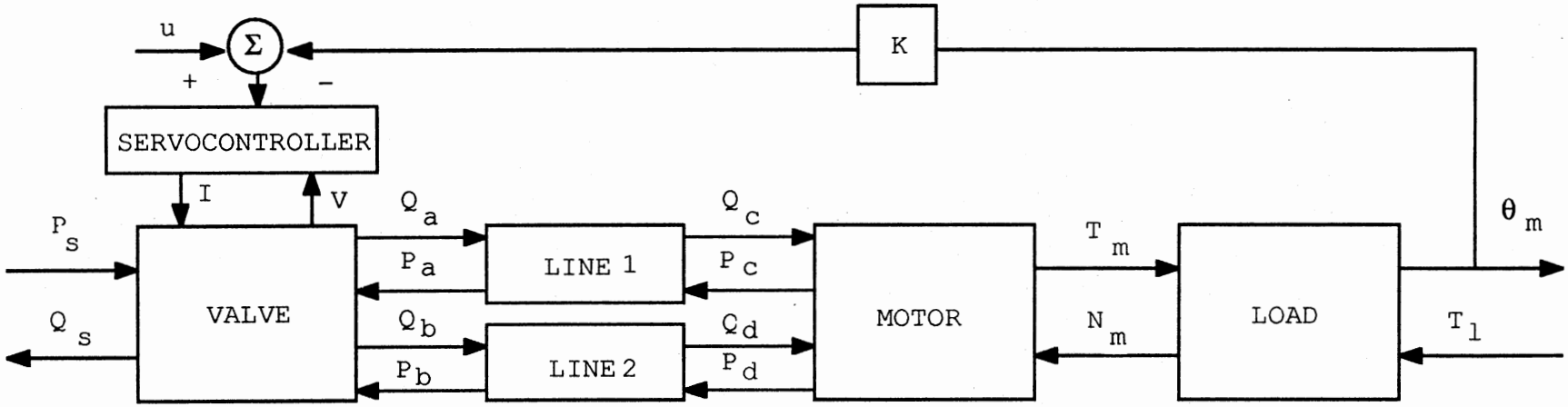


Figure 11. Multiport Representation of the System with Transmission Lines

the line kinematic viscosity, L the line length, c_0 the isentropic speed of sound in the fluid, and r_0 the inside radius of the line. Definitions for T_e, Y_s, and Z_s are as follows,

$$T_e = \frac{L}{c_0}$$

$$Y_s = \frac{\pi r_0^2}{\sqrt{\rho\beta}}$$

$$Z_s = \frac{1}{Y_s}$$

Figures 12 and 13 show plots developed by Gerlach (34) that can be used to calculate ω_{cn} and ζ_{cn} for particular value of the damping number. The zeroth order model values, ω_{c0} and ζ_{c0}, are determined using the n =0 curves.

Closed-Loop System Dynamic Behavior Using Initial Values of Parameters

Results of computer simulations for two different line lengths with equal inside radii are shown in Figure 14 and 15 respectively. The simulation program is listed in Appendix G. All parameters were identical to those used in the simulation for Example 1 (Figure 7) except the volume under compression was reduced to v = 2.5 in^3; this change was made to magnify the effects of the dynamics of the transmission lines.

Figure 14 shows the response for a line length of 200

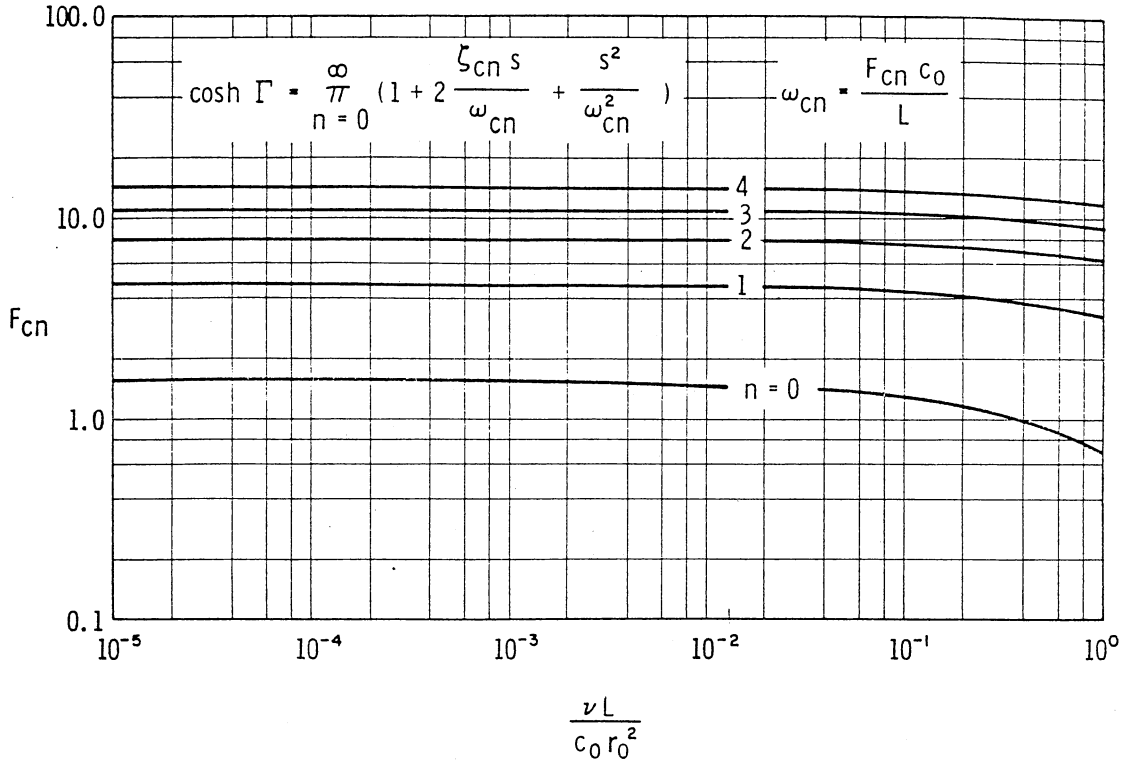


Figure 12. Variation of the Approximate Model Parameter ω_{cn} with Damping Number (Ref. (34))

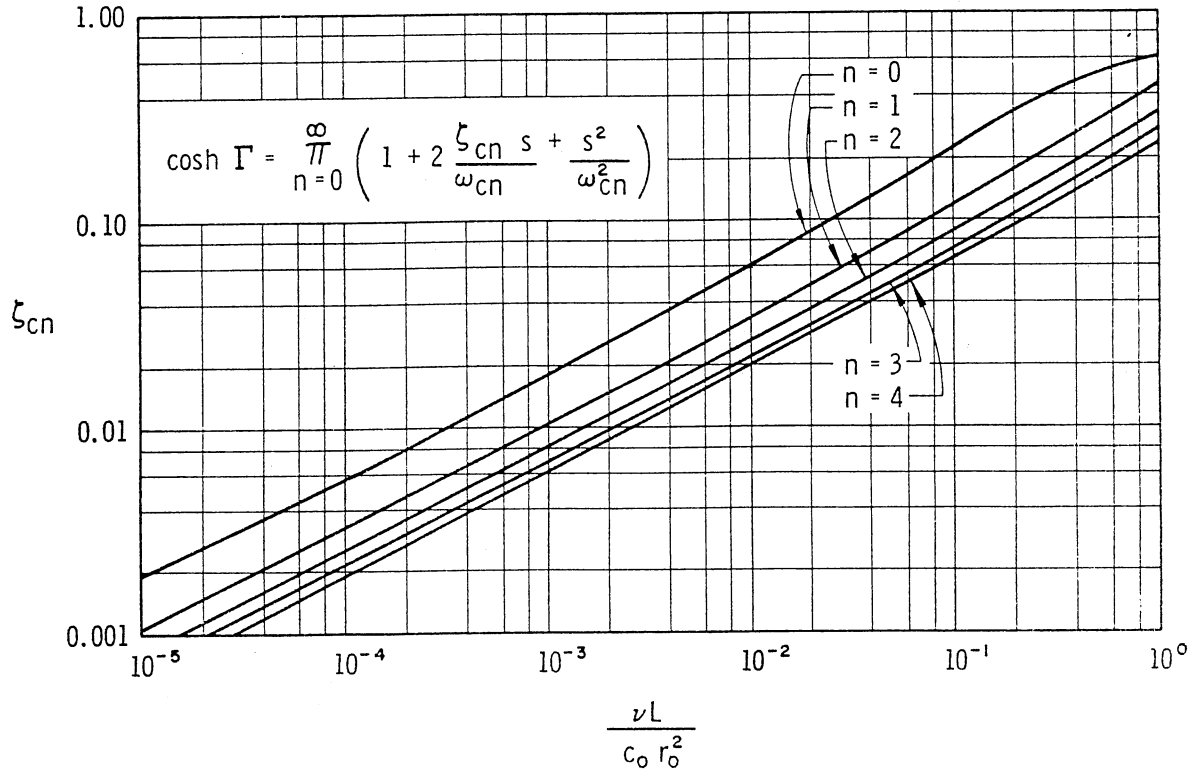


Figure 13. Variation of the Approximate Model Parameter ζ_{cn} with Damping Number (Ref. (34))

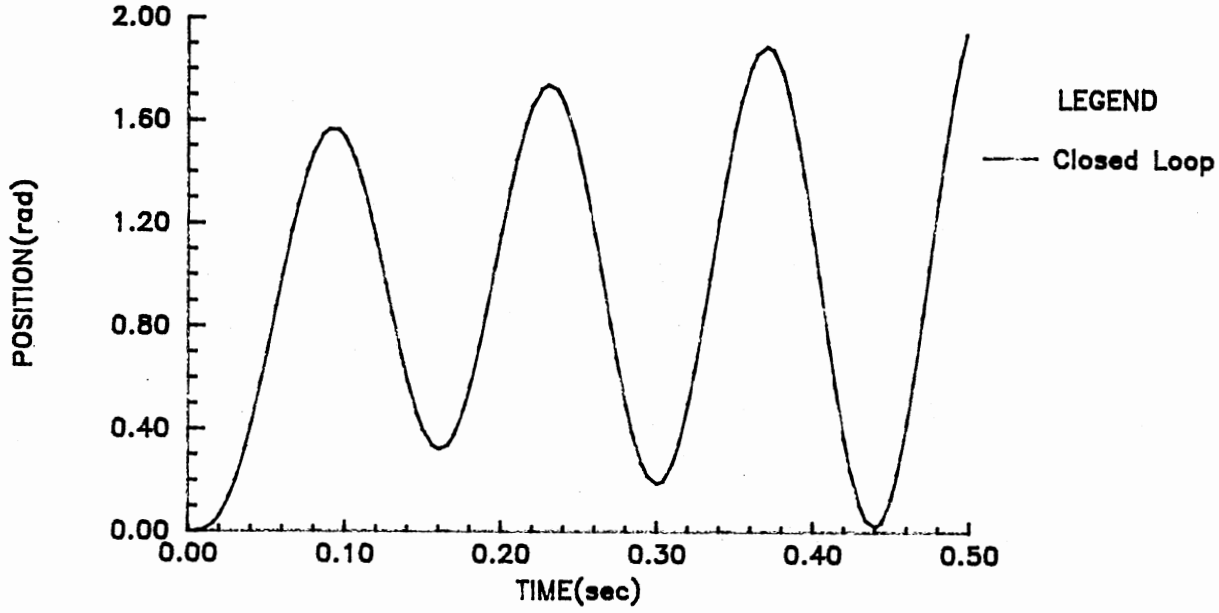


Figure 14. Response of the Position Feedback System to a Unit Step Input (Example 2; Initial Parameter Values; Without Pressure Feedback; L=200 in.)

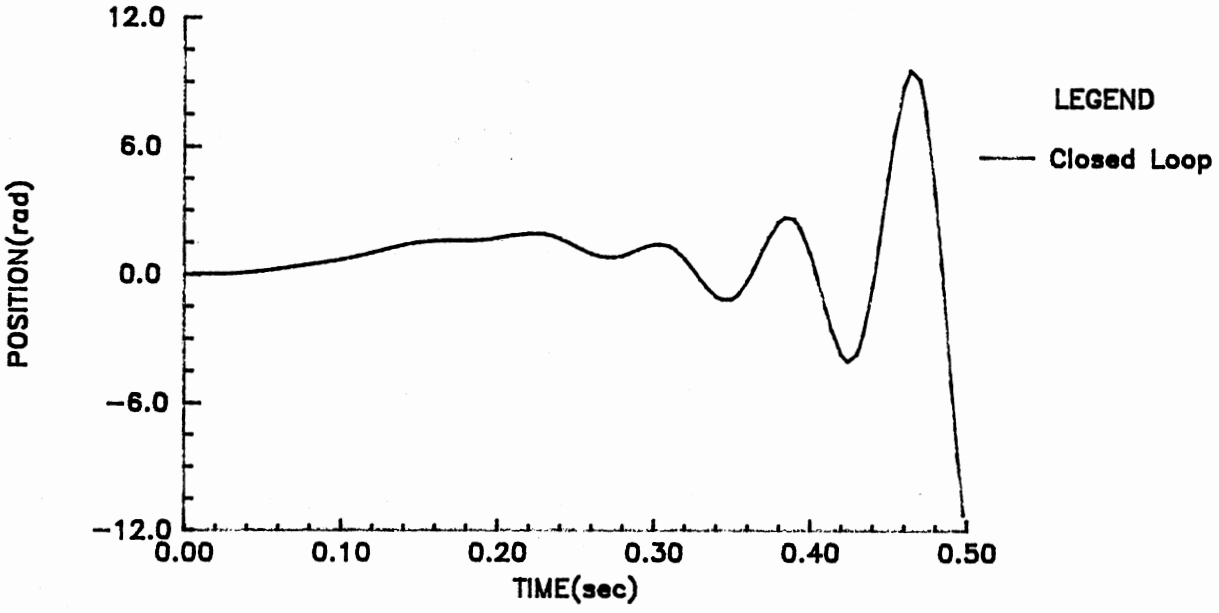


Figure 15. Response of the Position Feedback System to a Unit Step Input (Example 2; Initial Parameter Values; Without Pressure Feedback; L=1000 in.)

in. and Figure 15 shows the response for a line length of 1000 in. Both responses illustrate that the additional dynamics from a long line may result in an instability. The use of the optimization algorithm to determine parameters which produce stable operation is illustrated in the following sections.

Optimization of the Position Feedback System with Transmission Lines

The state and algebraic variables are redefined as follows,

$P_a = x_2$

$P_b = x_7$

$P_c = y_3$

$P_d = x_{10}$

$Q_a = y_1$

$Q_c = x_5$

$Q_d = y_2$

$I = y_4$

$N_m = x_{13}$

$\theta_m = x_{14}$

Combining Equations (5.27) through (5.30) and Equations (5.1) through (5.8) yields fourteen state equations:

$$\dot{x}_1 = x_2 \tag{5.31}$$

$$\dot{x}_2 = -2\omega_{c0}^2 x_1 - 2\zeta_{c0}\omega_{c0} x_2 + \omega_{c0}^2 x_3 + \omega_{c0}^2 Z_s T_e y_1 \tag{5.32}$$

$$\dot{x}_3 = x_2 + y_2 \tag{5.33}$$

$$\dot{x}_4 = x_5 \tag{5.34}$$

$$\dot{x}_5 = -2\omega_{c0}^2 x_4 - 2\zeta_{c0}\omega_{c0} x_5 + \omega_{c0}^2 x_6 - \omega_{c0}^2 Y_s T_e y_3 \tag{5.35}$$

$$\dot{x}_6 = x_5 + y_1 \tag{5.36}$$

$$\dot{x}_7 = x_8 \tag{5.37}$$

$$\dot{x}_8 = -\omega_{c0}^2 x_7 + \left(\frac{\omega_{c0}^2 Y_s T_e}{K_2} - 2\zeta_{c0}\omega_{c0}\right)x_8 + \omega_{c0}^2 y_2 + \frac{KK_1}{K_2 J}\left[c_d \mu D_m x_{13} - D_m(1-c_f)x_{12}\right] - \frac{2\zeta_{c0}\omega_{c0} KK_1}{K_2} x_{13} - \omega_{c0}^2 K_1 y_4 \tag{5.38}$$

$$\dot{x}_9 = x_{10} \tag{5.39}$$

$$\dot{x}_{10} = -2\omega_{c0}^2 x_9 - 2\zeta_{c0}\omega_{c0} x_{10} + \omega_{c0}^2 x_{11} + \omega_{c0}^2 Z_s T_e y_2 \tag{5.40}$$

$$\dot{x}_{11} = x_{10} + x_7 \tag{5.41}$$

$$\dot{x}_{12} = 2\frac{\beta}{v}\left(-D_m x_{13} - c_s \frac{D_m}{\mu} x_{12} + x_5\right) \tag{5.42}$$

$$\dot{x}_{13} = \frac{1}{J}\left(-c_d \mu D_m x_{13} + D_m(1-c_f)x_{12}\right) \tag{5.43}$$

$$\dot{x}_{14} = x_{13} \tag{5.44}$$

and four algebraic equations:

$$y_1 = K_1 y_4 + K_2 x_2 \tag{5.45}$$

$$y_2 = x_5 \tag{5.46}$$

$$y_3 = x_{10} + x_{12} \tag{5.47}$$

$$y_4 = u - K x_{14} \tag{5.48}$$

A combined function can be formed as follows,

$$H = -\left| x_{14} - R(t) \right| + \sum_{i=1}^{14} \lambda_i \dot{x}_i + \sum_{j=1}^{4} \psi_j \dot{y}_j \tag{5.49}$$

where $\dot{x}_i$ are given in Equations (5.31) through (5.44).

To obtain the expressions for $\dot{y}_j$, the Implicit Method was applied to Equations (5.45) through (5.48). The additional state equations are given below:

$$\dot{y}_1 = -K K_1 x_{13} + K_2 (-2\omega_{c0}^2 x_1 - 2\zeta_{c0}\omega_{c0} x_2 + \omega_{c0}^2 x_3 + \omega_{c0}^2 Z_s T_e y_1) \tag{5.50}$$

$$\dot{y}_2 = (-2\omega_{c0}^2 x_4 - 2\zeta_{c0}\omega_{c0} x_5 + \omega_{c0}^2 x_6 - \omega_{c0}^2 Y_s T_e y_3) \tag{5.51}$$

$$\dot{y}_3 = (-2\omega_{c0}^2 x_9 - 2\zeta_{c0}\omega_{c0} x_{10} + \omega_{c0}^2 x_{11} + \omega_{c0}^2 Z_s T_e y_2) + 2\frac{\beta}{v}\left(-D_m x_{13} - c_s \frac{D_m}{\mu} x_{12} + x_5\right) \tag{5.52}$$

$$\dot{y}_4 = -K\dot{x}_{14} = -K x_{13} \tag{5.53}$$

The eighteen total state equations, Equations (5.31) through (5.44) and (5.50) through (5.53) can be substituted into the combined function, Equation (5.49). By differentiating the combined function according to Equations (3.29) and (3.30), the necessary conditions or adjoint equations are obtained as follows,

$$\frac{d\lambda_1}{dt} = \lambda_2(2\omega_{c0}^2) + \psi_1(2K_2\omega_{c0}^2)$$

$$\frac{d\lambda_2}{dt} = -\lambda_1 + \lambda_2(2\zeta_{c0}\omega_{c0}) - \lambda_3 + \psi_1(2K_2\zeta_{c0}\omega_{c0})$$

$$\frac{d\lambda_3}{dt} = -\lambda_2(\omega_{c0}^2) - \psi_1(K_2\omega_{c0}^2)$$

$$\frac{d\lambda_4}{dt} = \lambda_5(2\omega_{c0}^2) + \psi_2(2\omega_{c0}^2)$$

$$\frac{d\lambda_5}{dt} = -\lambda_4 + \lambda_5(2\zeta_{c0}\omega_{c0}) - \lambda_6 - \lambda_{12}\left(2\frac{\beta}{\mu}\right) + \psi_2(2\zeta_{c0}\omega_{c0}) - \psi_3\left(2\frac{\beta}{v}\right)$$

$$\frac{d\lambda_6}{dt} = -\lambda_5(\omega_{c0}^2) - \psi_2(\omega_{c0}^2)$$

$$\frac{d\lambda_7}{dt} = \lambda_8(\omega_{c0}^2) - \lambda_{11}$$

$$\frac{d\lambda_8}{dt} = -\lambda_7 - \lambda_8\left(\frac{\omega_{c0}^2 Y_s T_e}{K_2} - 2\zeta_{c0}\omega_{c0}\right)$$

$$\frac{d\lambda_9}{dt} = \lambda_{10}(2\omega_{c0}^2) + \psi_3(2\omega_{c0}^2)$$

$$\frac{d\lambda_{10}}{dt} = -\lambda_9 + \lambda_{10}(2\zeta_{c0}\omega_{c0}) - \lambda_{11} + \psi_3(2\zeta_{c0}\omega_{c0})$$

$$\frac{d\lambda_{11}}{dt} = -\lambda_{10}(\omega_{c0}^2) - \psi_3(\omega_{c0}^2)$$

$$\frac{d\lambda_{12}}{dt} = \lambda_8\left\{\frac{KK_1}{K_2}\frac{D_m(1-c_f)}{J}\right\} + \lambda_{12}\left(2\frac{\beta}{v}c_s\frac{D_m}{\mu}\right) - \lambda_{13}\left\{\frac{D_m(1-c_f)}{J}\right\} + \psi_3\left(2\frac{\beta}{v}c_s\frac{D_m}{\mu}\right)$$

$$\frac{d\lambda_{13}}{dt} = -\lambda_8\left\{\frac{KK_1}{K_2}c_d\mu D_m - \frac{2\zeta_{c0}\omega_{c0}KK_1}{K_2}\right\} + \lambda_{12}\left(2\frac{\beta}{v}D_m\right) + \lambda_{13}\left\{\frac{c_d\mu D_m}{J}\right\}$$

$$-\lambda_{14} + \psi_1(KK_1) + \psi_3\left(2\frac{\beta}{v}D_m\right) + \psi_4(K)$$

$$\frac{d\lambda_{14}}{dt} = 1 \qquad \text{for } x_{14} \geq R(t), \qquad \frac{d\lambda_{14}}{dt} = -1 \qquad \text{for } x_{14} \geq R(t)$$

$$\frac{d\psi_1}{dt} = -\lambda_2(\omega_{c0}^2 Z_s T_e) - \lambda_6 - \psi_1(K_2\omega_{c0}^2 Z_s T_e)$$

$$\frac{d\psi_2}{dt} = -\lambda_8(\omega_{c0}^2) - \lambda_{10}(\omega_{c0}^2 Z_s T_e) - \psi_3(\omega_{c0}^2 Z_s T_e)$$

$$\frac{d\psi_3}{dt} = -\lambda_3 + \lambda_5(\omega_{c0}^2 Y_s T_e) + \psi_2(\omega_{c0}^2 Y_s T_e)$$

$$\frac{d\psi_4}{dt} = \lambda_8(\omega_{c0}^2 K_1)$$

For this example, following parameters are chosen for optimization:

$p_1 = r_0$ (inside radius of the line)

$p_2 = D_m$ (motor displacement)

$p_3 = K_1$ (valve coefficient)

Differentiating the combined function, Equation (5.49), with respect to each parameter results in three gradient vectors as shown below:

$$\frac{\partial H}{\partial p_1} = \lambda_2\left\{-4\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_1 - 2\frac{d(\zeta_{c0}\omega_{co})}{dp_1}x_2 + 2\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_3 + \frac{d(\omega_{c0}^2 Z_s T_e)}{dp_1}y_1\right\}$$

$$+\lambda_2\left\{-4\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_4 - 2\frac{d(\zeta_{c0}\omega_{co})}{dp_1}x_5 + 2\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_6 - \frac{d(\omega_{c0}^2 Y_s T_e)}{dp_1}y_3\right\}$$

$$+\lambda_8\left[-2\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_7+\left\{\frac{d(\omega_{c0}^2 Y_s T_e)}{dp_1}\frac{1}{K_2}-2\frac{d(\zeta_{c0}\omega_{co})}{dp_1}\right\}x_8+2\omega_{c0}\frac{d\omega_{c0}}{dp_1}y_2\right.$$

$$\left.-2\frac{d(\zeta_{c0}\omega_{co})}{dp_1}\frac{KK_1}{K_2}x_{13}-2\omega_{c0}\frac{d\omega_{c0}}{dp_1}K_1 y_4\right]$$

$$+\lambda_{10}\left\{-4\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_9-2\frac{d(\zeta_{c0}\omega_{co})}{dp_1}x_{10}+2\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_{11}+\frac{d(\omega_{c0}^2 Z_s T_e)}{dp_1}y_2\right\}$$

$$+\psi_1 K_2\left\{-4\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_1-2\frac{d(\zeta_{c0}\omega_{co})}{dp_1}x_2+2\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_3+\frac{d(\omega_{c0}^2 Z_s T_e)}{dp_1}y_1\right\}$$

$$+\psi_2\left\{-4\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_4-2\frac{d(\zeta_{c0}\omega_{co})}{dp_1}x_5+2\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_6-\frac{d(\omega_{c0}^2 Y_s T_e)}{dp_1}y_3\right\}$$

$$+\psi_3\left\{-4\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_9-2\frac{d(\zeta_{c0}\omega_{co})}{dp_1}x_{10}+2\omega_{c0}\frac{d\omega_{c0}}{dp_1}x_{11}+\frac{d(\omega_{c0}^2 Z_s T_e)}{dp_1}y_2\right\},$$

(5.54)

$$\frac{\partial H}{\partial p_2}=\lambda_8\frac{Kp_3}{K_2 J}\left\{c_d\mu x_{13}-(1-c_f)x_{12}\right\}-\lambda_{12}2\frac{\beta}{v}\left(x_{13}+\frac{c_s}{\mu}x_{12}\right)$$

$$+\lambda_{13}\frac{1}{J}\left\{-c_d\mu x_{13}+(1-c_f)x_{12}\right\}-\psi_3 2\frac{\beta}{v}\left(x_{13}+\frac{c_s}{\mu}x_{12}\right)$$

$$-\psi_3\frac{\beta}{v}\left(x_{13}+\frac{c_s}{\mu}x_{12}\right), \qquad (5.55)$$

$$\frac{\partial H}{\partial p_3}=\lambda_8\left[\frac{Kp_2}{K_2 J}\left\{c_d\mu x_{13}-(1-c_f)x_{12}\right\}-2\frac{\zeta_{c0}\omega_{c0}K}{K_2}x_{13}-\omega_{c0}^2 y_4\right]-\psi_1(Kx_{13})$$

(5.56)

The parameters ω_{c0} and ζ_{c0} can be represented by the following expressions obtained from fitting curves to the plots in Figures 12 and 13:

$$\zeta_{\omega} = 0.7138\left(\frac{\nu L}{c_0 r_0^2}\right)^{0.5273}$$

$$\omega_{\omega} = c_0 \frac{1.3978\, e^{-0.8403\left(\frac{\nu L}{c_0 r_0^2}\right)}}{L}$$

Tabulated results of continuous optimization are shown in Tables IV and V. The optimization program for this case is listed in Appendix J. Responses of the system to a unit step input were obtained by simulation using the continuous optimization parameter values. Figures 16 and 17 are the responses for the two line lengths considered. Both responses fit the desired response in a least squares sense, but both indicate poor dynamic performance in the sense of degree of stability. Improvement in degree of stability by means of pressure feedback is illustrated in the following section.

Optimization of the Position Control System with Transmission Lines and Pressure Feedback

Adding the pressure feedback affects some equations in the previous section. Equation (5.48) is changed to

$$y_4 = u - K x_{14} - K_p x_{12} \tag{5.57}$$

Applying the Implicit Method to Equation (5.57) gives

$$\dot{y}_4 = -K \dot{x}_{14} - K_p \dot{x}_{12}$$

or

TABLE IV

RESULTS OF CONTINUOUS OPTIMIZATION FOR EXAMPLE 2
WITHOUT PRESSURE FEEDBACK (L=200 in.)

iteration	p_1	p_2	p_3	error
1	0.15	0.015	0.25	3.852
3	0.136	0.0163	0.228	0.705
5	0.131	0.0167	0.222	0.408
7	0.128	0.0169	0.219	0.289
9	0.126	0.0170	0.216	0.238
11	0.124	0.0172	0.214	0.196
13	0.122	0.0173	0.213	0.167
15	0.120	0.0173	0.211	0.147
17	0.118	0.0175	0.209	0.121

TABLE V

RESULTS OF CONTINUOUS OPTIMIZATION FOR EXAMPLE 2
WITHOUT PRESSURE FEEDBACK (L=1000 in.)

iteration	p_1	p_2	p_3	error
1	0.15	0.015	0.25	2.576
3	0.130	0.0169	0.226	0.888
5	0.119	0.0178	0.217	0.491
7	0.110	0.0183	0.214	0.301
9	0.104	0.0186	0.213	0.212
11	0.099	0.0188	0.212	0.164
13	0.095	0.0189	0.211	0.134

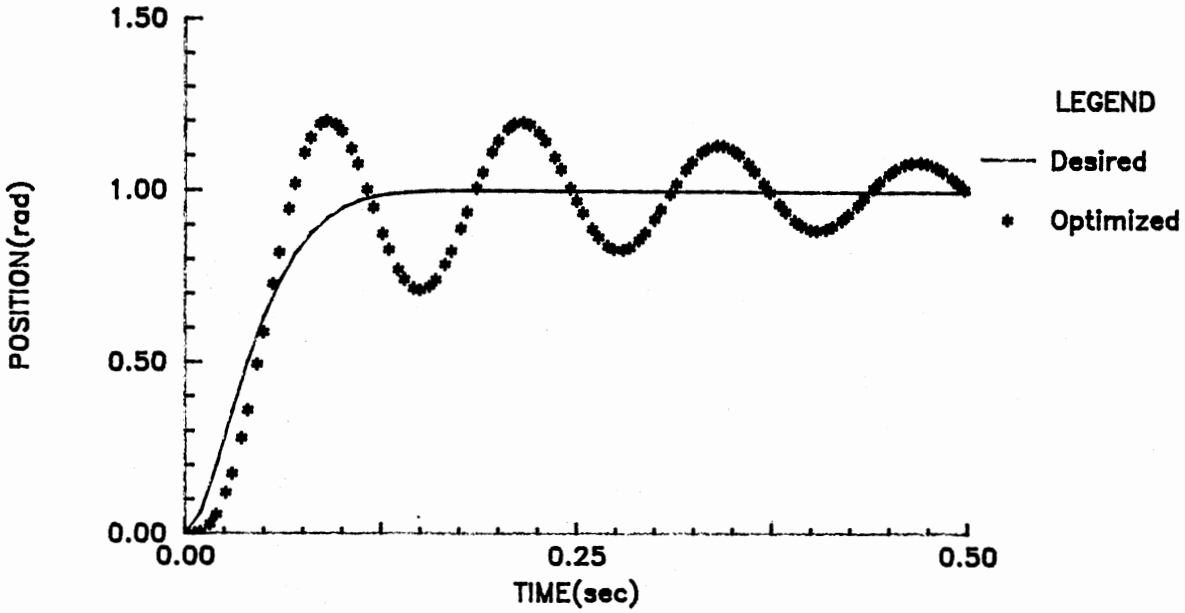


Figure 16. Response of the Position Feedback System to a Unit Step Input (Example 2; Optimized Continuous Parameter Values; Without Pressure Feedback; L=200 in.)

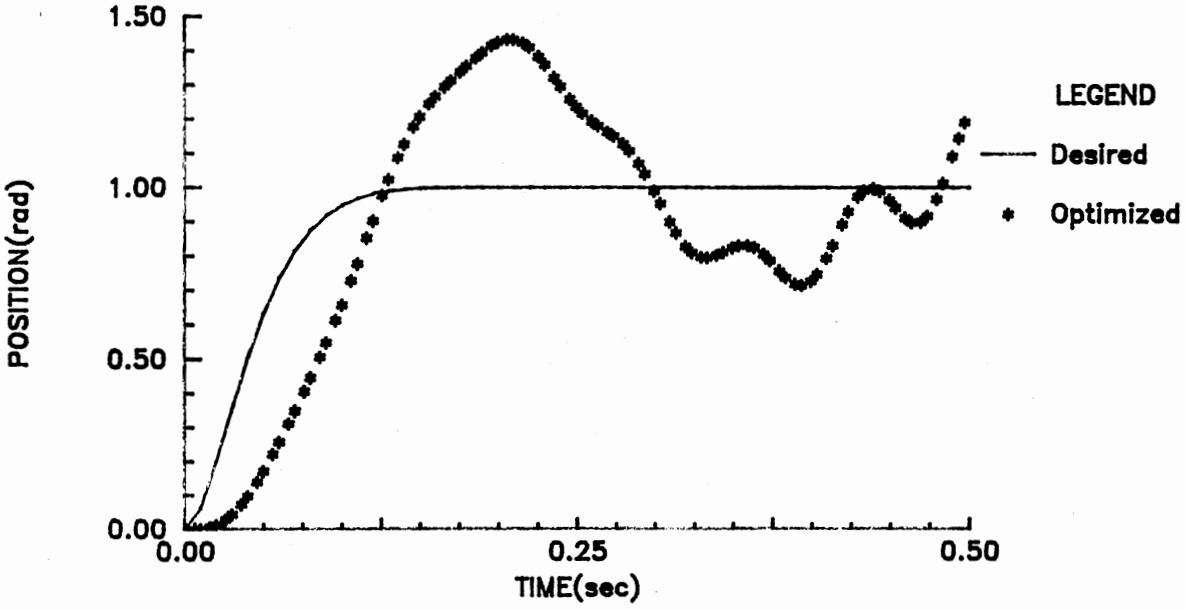


Figure 17. Response of the Position Feedback System to a Unit Step Input (Example 2; Optimized Continuous Parameter Values; Without Pressure Feedback; L=1000 in.)

$$\dot{y}_4 = Kx_{13} - K_p 2\frac{\beta}{v}\left(-D_m x_{13} - c_s\frac{D_m}{\mu}x_{12} + x_5\right)$$

Several adjoint equations also are changed as follows,

$$\frac{d\lambda_5}{dt} = -\lambda_4 + \lambda_5(2\zeta_{c0}\omega_{c0}) - \lambda_6 - \lambda_{12}\left(2\frac{\beta}{\mu}\right)$$

$$+ \psi_2(2\zeta_{c0}\omega_{c0}) - \psi_3\left(2\frac{\beta}{v}\right) + \psi_4\left(2K_p\frac{\beta}{v}\right)$$

$$\frac{d\lambda_{12}}{dt} = \lambda_8\left\{\frac{KK_1}{K_2}\frac{D_m(1-c_f)}{J}\right\} + \lambda_{12}\left(2\frac{\beta}{v}c_s\frac{D_m}{\mu}\right) - \lambda_{13}\left\{\frac{D_m(1-c_f)}{J}\right\}$$

$$+ \psi_3\left(2\frac{\beta}{v}c_s\frac{D_m}{\mu}\right) - \psi_4\left(2K_p\frac{\beta}{v}c_s\frac{D_m}{\mu}\right)$$

$$\frac{d\lambda_{13}}{dt} = -\lambda_8\left\{\frac{KK_1}{K_2}c_d\mu D_m - \frac{2\zeta_{c0}\omega_{c0}KK_1}{K_2}\right\} + \lambda_{12}\left(2\frac{\beta}{v}D_m\right) + \lambda_{13}\left\{\frac{c_d\mu D_m}{J}\right\}$$

$$-\lambda_{14} + \psi_1(KK_1) + \psi_3\left(2\frac{\beta}{v}D_m\right) + \psi_4\left(K - 2K_p\frac{\beta}{v}D_m\right)$$

The other adjoint equations remain the same.

The pressure feedback gain K_p is an appropriate parameter to be optimized; here $K_p = p_4$. Then, Equation (5.55) has the following additional term:

$$2\psi_4\left(p_4\frac{\beta}{v}\right)\left(x_{13} + \frac{c_s}{\mu}x_{12}\right) \text{ for } \frac{\partial H}{\partial p_2}.$$

Also, a new gradient vector is formed for the additional parameter as follows,

$$\frac{\partial H}{\partial p_4} = \psi_4 2\frac{\beta}{v}\left(D_m x_{13} + c_s\frac{D_m}{\mu}x_{12} - x_5\right).$$

Tabulated results for the continuous optimization are shown in Tables VI and VII. The responses shown in Figures

TABLE VI

RESULTS OF CONTINUOUS OPTIMIZATION FOR EXAMPLE 2
WITH PRESSURE FEEDBACK (L=200 in.)

iteration	p_1	p_2	p_3	p_4	error
1	0.15	0.015	0.25	0.005	1.425
3	0.1337	0.0166	0.226	0.00549	0.341
5	0.1269	0.0171	0.218	0.00563	0.196
7	0.1223	0.0174	0.214	0.00571	0.138
9	0.1187	0.0176	0.211	0.00576	0.108
11	0.1158	0.0177	0.209	0.00581	0.087
13	0.1132	0.0178	0.207	0.00584	0.074
15	0.1110	0.0179	0.206	0.00587	0.065
17	0.1090	0.0180	0.204	0.00589	0.057
19	0.1073	0.0181	0.204	0.00591	0.051
21	0.1060	0.0181	0.203	0.00594	0.046
23	0.1040	0.0182	0.198	0.00600	0.041

TABLE VII

RESULTS OF CONTINUOUS OPTIMIZATION FOR EXAMPLE 2
WITH PRESSURE FEEDBACK (L=1000 in.)

iteration	p_1	p_2	p_3	p_4	error
1	0.15	0.015	0.25	0.005	1.129
2	0.140	0.0161	0.233	0.00535	0.724
3	0.137	0.0162	0.114	0.00542	0.244
4	0.136	0.0163	0.053	0.00543	0.137

18 and 19 were determined through simulation using the continuous optimization results. The optimization program for this case is listed in Appendix K.

To make the example more interesting, both p_2 and p_3 are treated as discrete parameters. The available discrete values for the parameters are shown in Table VIII. Application of the branch and bound method yielded the results shown in Figure 20 for L=200 inches. By comparison of the errors, the second branch in Figure 20 is found to yield the new set of optimal parameters listed below:

$p_1 = 0.0885$ in

$p_2 = 0.01618$ in^3/rad

$p_3 = 0.15$ in^3/sec ma

$p_4 = 0.00539$ in^2 ma/lbf

These results differ somewhat from the continuous optimization results in Table VI. The system response obtained by simulation using the new parameters is compared with the response based on the continuous optimization parameters in Figure 21. The responses differ only slightly even though the parameters differ by a significant amount.

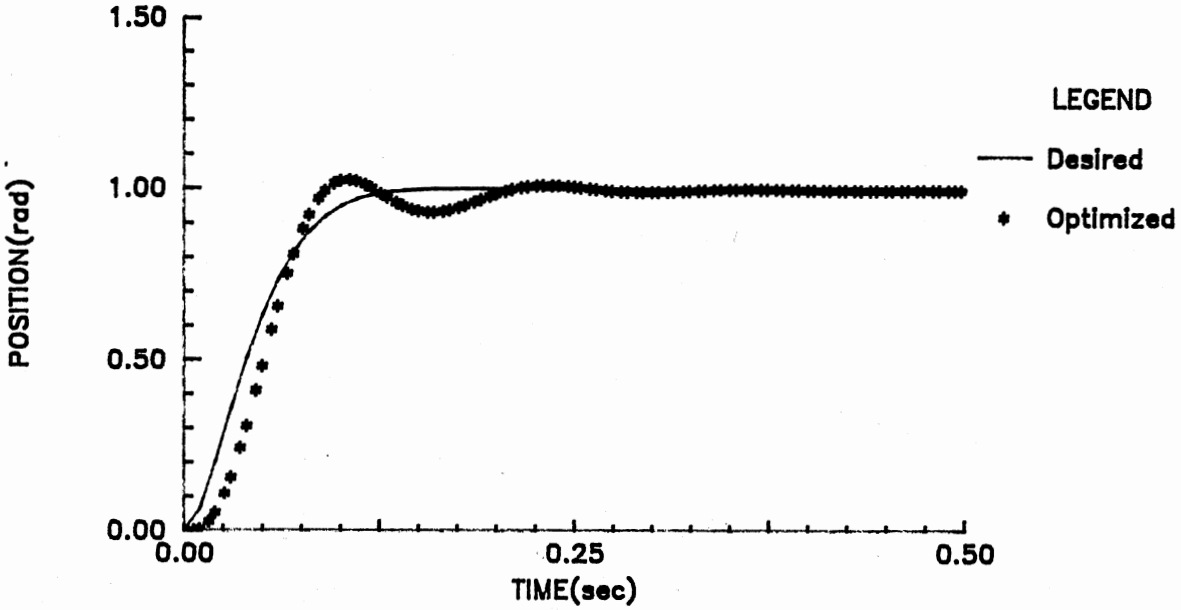


Figure 18. Response of the Position Feedback System to a Unit Step Input (Example 2; Optimized Continuous Parameter Values; With Pressure Feedback; L=200 in.)

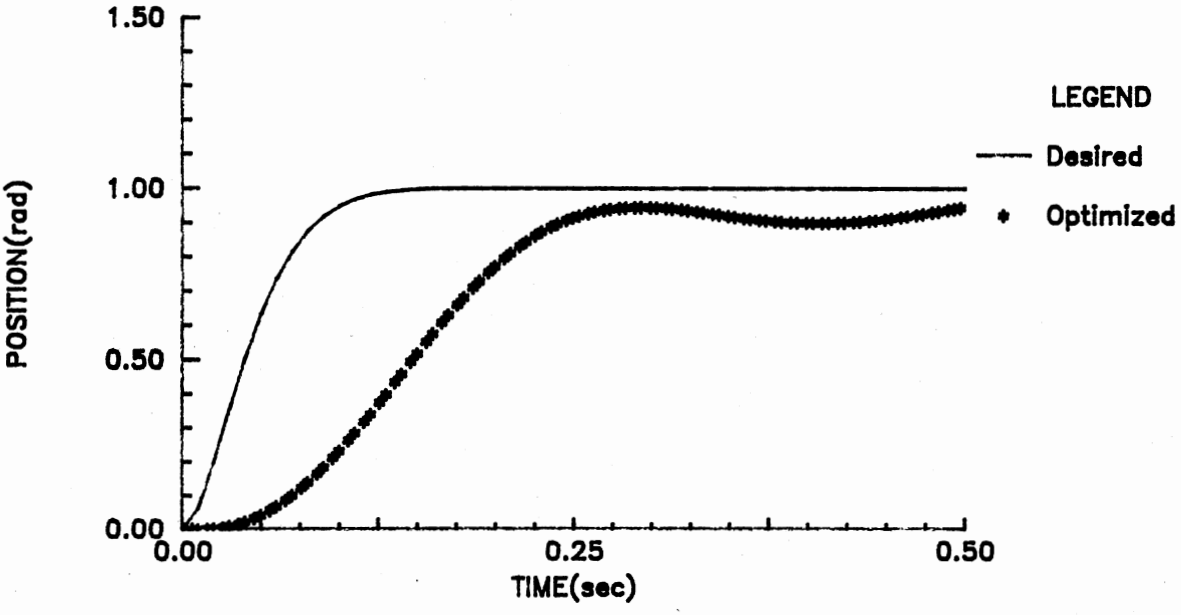
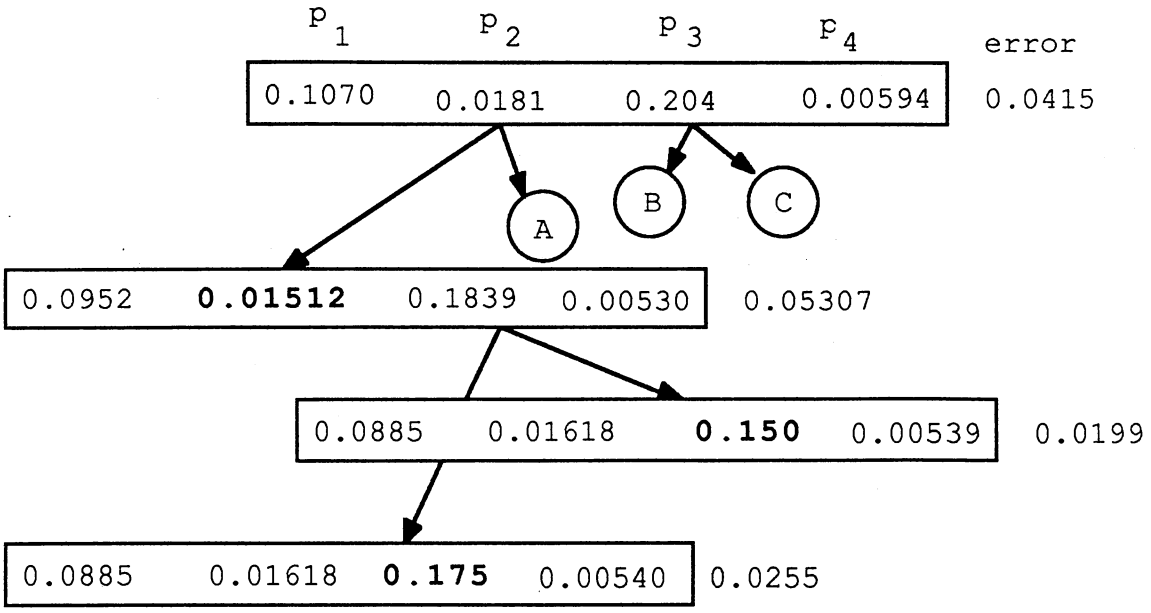


Figure 19. Response of the Position Feedback System to a Unit Step Input (Example 2; Optimized Continuous Parameter Values; With Pressure Feedback; L=1000 in.)

TABLE VIII

DISCRETE PARAMETER VALUES

Parameter	Discrete Values						
Motor Displacement, $p_2 = D_m$	0.009549	0.01512	0.030	0.05841	0.09549		
Valve Coefficient, $p_3 = K_1$	0.1	0.125	0.15	0.175	0.2	0.225	0.25



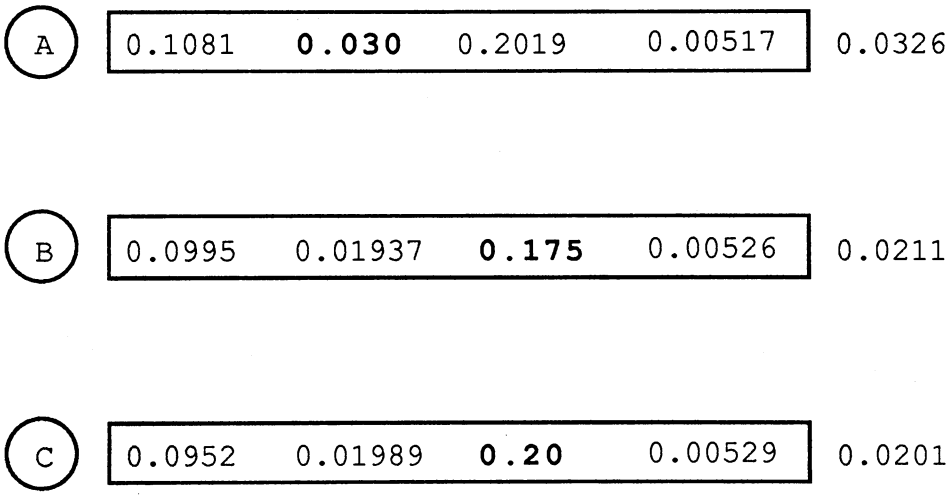


Figure 20. Branch-and-Bound Tree for Discrete Parameters

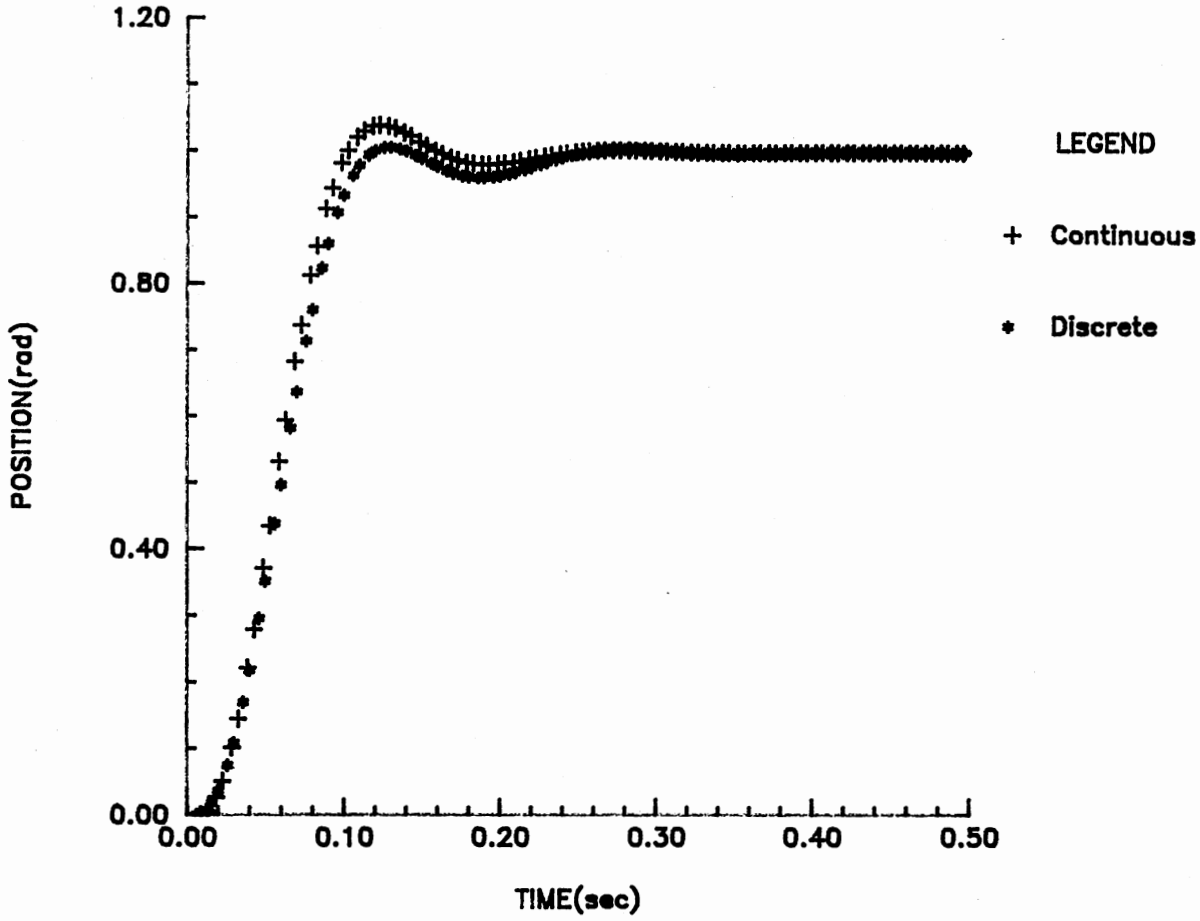


Figure 21. Comparison between the Response based on Optimized Continuous Parameters and the Response based on Optimized Discrete Parameters

CHAPTER VI

SUMMARY AND RECOMMENDATIONS

Summary

The design parameters of hydraulic systems govern the static and dynamic performance characteristics. A desired performance can be approached by selecting optimal parameters. A new functional optimization algorithm was developed in this study which has the following principle features:

1) The algorithm can solve the parameter optimization problem for a dynamic system which is described by a coupled set of state equations and algebraic equations.
2) The variational approach as modified for the algorithm allows more efficient optimization than an existing algebraic optimization method.
3) The use of the conjugate gradient technique speeds up convergence compared to that which would result with a simple gradient method.
4) The algorithm can optimize the discrete parameters that exist for real, standard, off-the-shelf, components.

The algorithm was applied to two examples, each with two levels of complexity, to illustrate the potential of the optimization method in the design of relatively complex hydraulic control systems. FORTRAN programs were written for the example problems. (Appendices H through K). The overall procedure used to solve the examples is diagrammed in Figure 22.

A limitation of the optimization algorithm is that linearization of nonlinear algebraic equations is necessary if the variables of the nonlinear algebraic equations are linked with the differential equations in such a manner that the nonlinearity prevents the differential equations from being transformed to state equations. Other limitation is that the 'optimized parameters' may not be the best parameters associated with a 'global optimum'. However, the local optimum can be a good solution.

The application of the optimization algorithm is not restricted to hydraulic systems. It can be used as a design tool for other dynamic systems which are modeled by coupled sets of state and algebraic equations, such as pneumatic systems, electromechanical systems, etc.

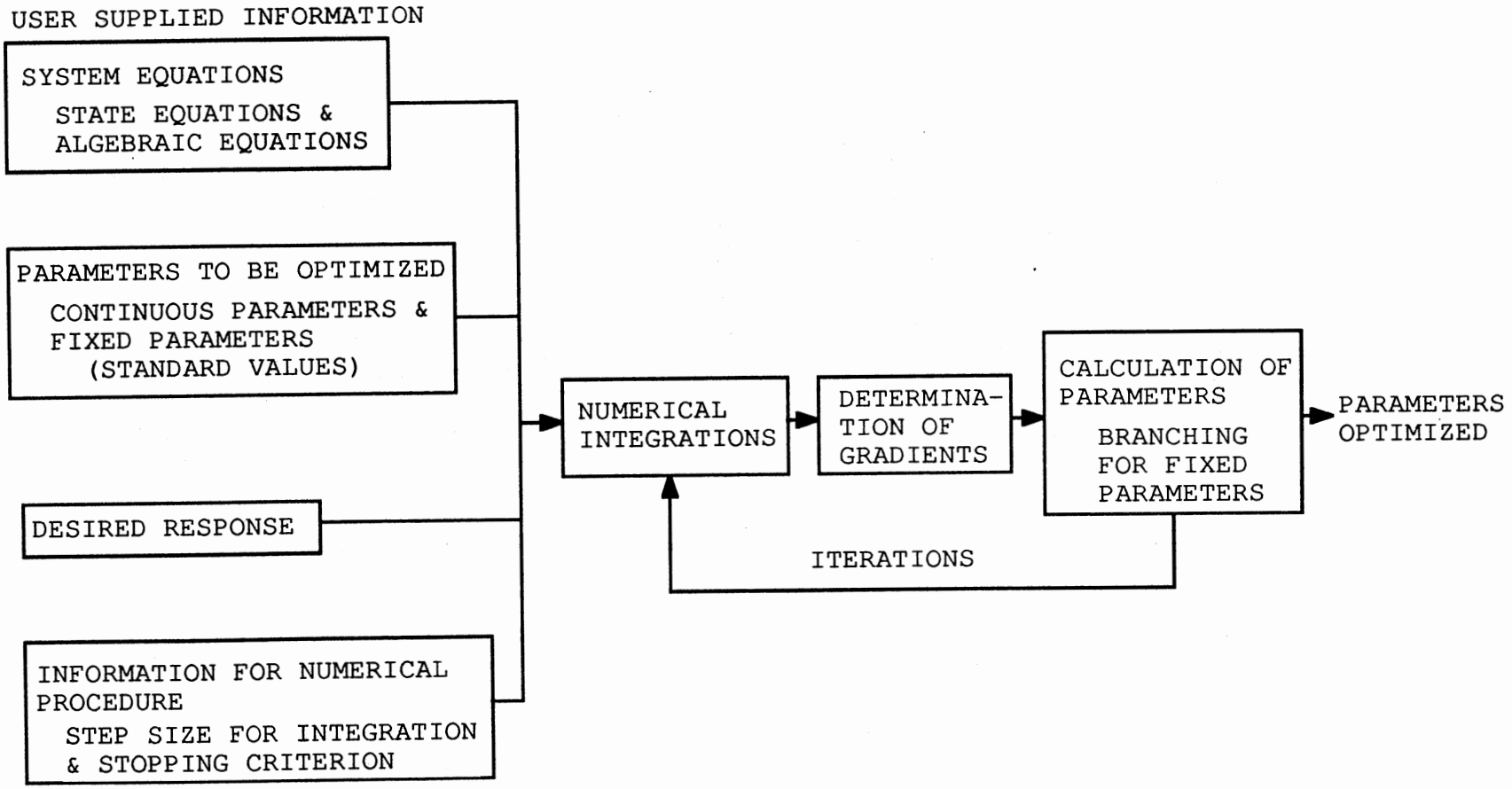


Figure 22. Overall Procedure of the Example Program

Recommendations for the Further Study

Following are recommendations for further study:

1. Develop a generalized computer code which incorporates the optimization algorithm. Ideally, this code would include a 'virtual front end' which limits the user input to system topology or component equations, initial values of the parameters, and the desired response. Potentially, a symbolic manipulation scheme could be used to automatically formulate the adjoint equations.
2. Develop an extended component library including thermal response.
3. Develop a method for finding the global minimum within the parameter space.
4. Perform experimental studies to verify the algorithm.

A SELECTED BIBLIOGRAPHY

(1) Smith, C. K. "Digital Simulation of Complex Hydraulic Systems Using Multiport Component Models." Unpubl. Ph.D. Diss., Oklahoma State University, 1975.

(2) Ebbesen, L. "Digital Computer Simulation of the Mechanical and Thermal Response of Complex Hydraulic Systems." Unpubl. Ph.D. Diss., Oklahoma State University, 1976.

(3) Dransfield, P. and R. Labrooy. "Designing Hydraulic Control Systems with Optimal Response." Annual Engineering Conference, Aust. (1976), 435-440.

(4) Ahmed, N. U. and N. D. Georganas. "On Optimal Parameter Selection." IEEE Transactions on Automatic Control, 18 (1973), 313-314.

(5) Pontryagin, Boltyanskii, Gamkrelitze, and Mischenko. The Mathematical Theory of Optimal Processes. New York: Wiley, 1962.

(6) Georganas, N. D. "Optimal Parameter Selection by Imbedding Techniques." IEEE Transactions on Automatic Control, 20 (1975), 166-167.

(7) Boltyanskii, V. G. Mathematical Methods of Optimal Control. New York: Holt, Rinehart and Winston, 1971.

(8) Ahmed, N. U. "A simple Gradient Algorithm for Least Squares Estimation of System Parameters." International Journal of Systems Science, 7 (1976), 673-677.

(9) Teo, K. L. and E. J. Moore. "On Directional Derivative Methods for Solving Optimal Parameter Selection Problems." International Journal of Systems Science, 9 (1978), 1029-1041.

(10) Dolezal, J. "Direct Method for Parameter Optimization." International Journal of Systems Science, 11 (1980), 337-343.

(11) Orurk, I. A., L. A. Osipuv, and L. G. Petukhov. "Parameter Optimization of Nonlinear Control Systems by Method of Orthogonal Projections." Automatic Remote Control, 42 (1981), 1459-1467.

(12) Gopalsami, N. and C. K. Sanathanan. "Satisfactory Solutions Approach to Parameter Optimization of Dynamic Systems with Vector Performance Index." Journal of Optimization Theory and Applications, 47 (1985), 301-319.

(13) Desantis, R. M. and W. A. Porter. "Optimization Problem in Partially Ordered Hillbert Resolution Spaces." International Journal of Control, 36 (1982), 875-883.

(14) Hsieh, C. C. and J. S. Arora. "Design Sensitivity Analysis and Optimization of Dynamic Response." Computer Methods in Applied Mechanics and Engineering, 43 (1984), 195-219.

(15) Masri, S. F., G. A. Bekey, and F. B. Safford. "A Global Optimization Algorithm Using Adaptive Random Search." Applied Mathematics and Computation, 7 (1980), 353-375.

(16) Pappas, M. "An Improved Direct Search Numerical Optimization Procedure." Computers and Structures, 11 (1980).

(17) Gear, C. W. "Simultaneous Numerical Solution of Differential-Algebraic Equations." IEEE Transactions on Circuit Theory, 18 (1971), 89-95.

(18) Luenberger, D. G. Linear and Nonlinear Programming. Menlo Park: Addison-Wesley, 1984.

(19) Lunderstadt, R. "Transportation Systems with Optimal Parameters." Problems of Control and Information Theory, 5 (1976), 109-116.

(20) Ortega, J. M. and W. G. Poole, Jr. Numerical Methods for Differential Equations. Marshfield: Pitman, 1981.

(21) Dransfield, P. and R. Labrooy. "Hydraulic Systems with Precision Reflexes." Machine Design, (May 1976), 106-109.

(22) Denn, M. M. Optimization by Variational Methods. New York: McGraw-Hill, 1969.

(23) Gottfried, B. and J. Weisman. Introduction to Optimization Theory. Englewood Cliffs: Prentice-Hall, 1973.

(24) McCausland, C. W. Introduction to Optimal Control. New York: Wiley, 1976.

(25) Merriam, C. W. Optimization and the Design of Feedback Control Systems. New York: McGraw-Hill, 1964.

(26) Carnahan, B., H. A. Luther, and J. O. Wilkes. Applied Numerical Methods. New York: Wiley, 1969.

(27) Hasdorf, L. Gradient Optimization and Nonlinear Control. New York: Wiley, 1976.

(28) Haug, E. J. and J. S. Arora. Applied Optimal Design. New York: Wiley, 1979.

(29) Miele, A. "Recent Advance in Gradient Algorithms for Optimal Control Problems." Journal of Optimization Theory and Applications, 17 (1975), 361-430.

(30) MAE 6453 Fluid Power Control II. Classnote, 1983.

(31) Merritt, H. Hydraulic Control Systems. New York: Wiley, 1966.

(32) Box, M. J. "Complex Algorithm." Classnote, 1983

(33) Gupta, Omprakash K, and A. Ravindran. "Nonlinear Integer Programming and Discrete Optimization." Journal of Mechanisms, Transmissions, and Automation in Design, 105(1983), 160-164.

(34) Gerlach, C. R. "Dynamics of Viscous Fluid Transmission Lines with Particular Emphasis on Higher Mode Propagation." Ph.D. Diss., Oklahoma State University, 1966.

APPENDIX A

THE ERROR INDEX

When the problem of optimization of a dynamic system is considered, there must be an error index which is the objective function to be minimized. The most common error indices used for the optimization problem are

$$I = \int_0^t |e| \, dt \tag{A.1}$$

$$I = \int_0^t e^2 \, dt \tag{A.2}$$

$$I = \int_0^t t|e| \, dt \tag{A.3}$$

The first index is a simple sum and the second one is a squared sum. The third one gives major weight to the final portion of the response. If the tradeoff in design involves the highest possible speed of response with an acceptable degree of stability, the most appropriate error index would be one that gives most weight to the transient region. This is normally the case. Then Equation (A.1) or (A.2) is most appropriate. Several example problems solved in this study showed that the solution does not converge as rapidly with

the squared sum index as with the simple sum index. Therefore, the simple sum index was chosen for this study.

The error e is the discrepancy between the desired and predicted response and can be written as

$$e = R_d(x,y,p) - R_p(t) \tag{A.4}$$

where R_d is the desired response and R_p is the predicted response. The desired response can be given in the form of an equation, (i.e., a function of time) or given as tabulated discrete data.

APPENDIX B

THE IMPLICIT METHOD

Given a dynamis system model in terms of the following coupled set of state and algebraic equations,

$$\dot{x}_i = G_i(x_1, x_2, \ldots, x_n, y_i, y_2, \ldots, y_m, t), \qquad i=1,2,\ldots,n \quad (B.1)$$

$$0 = F_j(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m, t), \qquad j=1,2,\ldots,m \quad (B.2)$$

y_j could be determined by an explicit analytical solution of Equation (B.2) were $y_j = f(t)$. However, in this case, $y_j = f(x_i,t)$, and y_j must be determined by a numerical procedure. A possible candidate is the Newton-Raphson method, but this method is inefficient compared to the Imlicit Method used by Smith (1).

The chain rule allows the derivatives of Equation (B.2) to be written as

$$\frac{dF_j}{dt} = \sum_{k=1}^{n} \frac{\partial F_j}{\partial x_k} \frac{dx_k}{dt} + \sum_{l=1}^{m} \frac{\partial F_j}{\partial y_l} \frac{dy_l}{dt} + \frac{\partial F_j}{\partial t} \qquad (B.3)$$

Since the right side of Equation (B.3) is equal to zero, it may be rearranged in vector matrix form to yield

$$\begin{bmatrix} \frac{\partial F_1}{\partial y_1} & \frac{\partial F_1}{\partial y_2} & \cdots & \frac{\partial F_1}{\partial y_m} \\ \frac{\partial F_2}{\partial y_1} & \frac{\partial F_2}{\partial y_2} & \cdots & \frac{\partial F_2}{\partial y_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial F_m}{\partial y_1} & \frac{\partial F_m}{\partial y_2} & \cdots & \frac{\partial F_m}{\partial y_m} \end{bmatrix} \begin{bmatrix} \frac{dy_1}{dt} \\ \frac{dy_2}{dt} \\ \vdots \\ \frac{dy_m}{dt} \end{bmatrix} = \begin{bmatrix} -\sum_{j=1}^{n} \frac{\partial F_1}{\partial x_j} \frac{dx_j}{dt} - \frac{\partial F_1}{\partial t} \\ -\sum_{j=1}^{n} \frac{\partial F_2}{\partial x_j} \frac{dx_j}{dt} - \frac{\partial F_2}{\partial t} \\ \vdots \\ -\sum_{j=1}^{n} \frac{\partial F_m}{\partial x_j} \frac{dx_j}{dt} - \frac{\partial F_m}{\partial t} \end{bmatrix} \qquad \text{(B.4)}$$

Equation (B.4) may be solved for dy_1/dt through dy_m/dt by substitution of the values of $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m$ into the elements of the matrices of each side in Equation (B.4) (The elements are represented in terms of the x, y's). By an integration method, the y's can be determined.

APPENDIX C

THE METHOD OF CONJUGATE GRADIENTS FOR ORDINARY OPTIMIZATION

In the simple gradient method, the search direction is determined by the current gradient vectors or partial derivatives of the objective function, (Equation (3.23)). However, in the conjugate gradient method, the search vectors are functions of both the current gradient vectors and the previous vectors. The conjugate gradient method is guaranteed to minimize a function with quadratic convergence.

Let an objective function have the form

$$y = y(x) \tag{C.1}$$

where x is the unknown variable vector. If an arbitrary point x_0 is chosen, the gradient vector ∇y_0 must be evaluated. If $c_0 = \pm \nabla y0$ (the plus sign corresponding to a maximization, the minus to a minimization) and the point x_{i+1} is determined by use of the following equation

$$x_{i+1} = x_i + \alpha_i c_i \tag{C.2}$$

where α_i is determined such that the objective function is minimized along c_i. The gradient vector ∇y_i is determined at x_i and a new vector c_i is determined by

$$c_{i+1} = \pm \nabla y_i + \beta_i c_i \qquad (C.3)$$

where β_i can be obtained simply by

$$\beta_i = \frac{\nabla y'_{i+1} \nabla y_{i+1}}{\nabla y'_i \nabla y_i} \qquad (C.4)$$

(primed vector indicates its transpose).

Implementation of the method of conjugate gradients requires only the partial derivatives of the objective function. Also, the method avoids the problem of oscillations near an optimum.

APPENDIX D

COMPARISON OF THE FUNCTIONAL OPTIMIZATION ALGORITHM OF THIS STUDY WITH "COMPLEX"

COMPLEX Algorithm

The COMPLEX algorithm is an algebraic (ordinary) optimization algorithm which is used for a class of problems where the objective function is not expressed as an integral (32). However, COMPLEX can also be used for optimization of a functional if the number of parameters are relatively small.

As an example of how COMPLEX works, consider a system where parameters p and q are used to optimize dynamic response. Somewhere on the p, q plane there is a point Z_{opt} that minimizes a chosen objective function.

The procedure is started with three random points representing particular values for p and q. COMPLEX evaluates the objective function for each of the three points, which can be done by a simulation. It replaces the worst point with a better point.

If Z_1 is found to be the worst point in the initial set Z_1, Z_2, and Z_3, the program reflects Z_1 through the centroid of the triangle formed by the three points to a new position Z_4. Because of the reflection, the new point Z_4 is likely

better than Z_1, and the centroid of the triangle Z_2, Z_3, and Z_4 is likely to be closer to Z_{opt} than the centroid of the initial set.

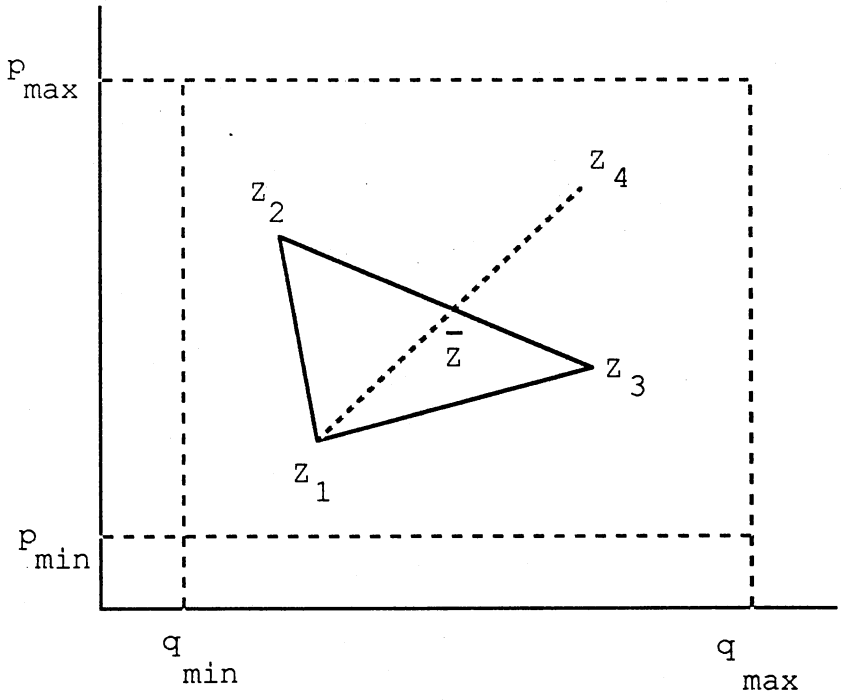


Figure 23. COMPLEX Procedure Approaching Minimum

The procedure illustrated in Figure 23 is repeated with the new set of points so that the set tends to converge to an optimal point.

Comparison of COMPLEX with the New Functional Optimization Algorithm on a Simple Case

Both methods were applied to the same electrohydraulic system as in Chapter V, but only two parameters were optimized here for programming convenience. They are

$p_1 = D_m$ (motor displacement)

$p_2 = v$ (volume under compression).

And the squared error index was used to enhance the difference. The results are shown in Tables IX and X. The convergence of the new functional optimization algorithm is better than COMPLEX since the error of the last iteration is much smaller. In addition, the new algorithm converged more than twice as fast in computational time. The reason for these results is that COMPLEX is an algebraic optimization algorithm which requires more simulation. If the order of a system is n and the number of parameters is l, COMPLEX requires integration of (l+1)n differential equations to form a initial set. It also requires integration of at least 2n differential equations at each iteration. However, the new algorithm presented in this study requires integration of 2n differential equation at each iteration, but does not require any integration to form an initial set.

TABLE IX

OPTIMIZATION BY THE NEW METHOD

iteration	p_1	p_2	error
1	.010000	50.000000	374.260468
2	.011400	43.000000	194.526077
3	.012015	36.154980	134.795990
4	.012378	29.109121	89.044830
5	.012630	20.973135	37.874691
6	.012757	13.089073	7.566950

TABLE X

OPTIMIZATOIN BY COMPLEX

iteration	p_1	p_2	error
1	.010000	50.000000	374.260468
2	.013750	56.250000	174.923508
3	.014750	51.250000	165.568893
4	.014750	51.250000	165.568893
5	.013938	44.062500	136.882629
6	.013938	44.062500	136.882629
7	.015031	32.968750	107.645050
8	.012530	30.570312	93.007507
9	.013545	13.330078	18.908537

APPENDIX E

THE RESULTS FOR DIFFERENT SETS OF INITIAL VALUES

Several different sets of initial values were given for the optimization of Example 1 discussed in Chapter V. The example system is the position control system with pressure feedback.

Following are Tables XI through XIII showing the results of continous optimization for three different sets of initial values. Each set resulted in the larger error than shown in Table III in Chapter V, which indicates that some set of initial values could converge to a local minimum.

TABLE XI

RESULTS OF CONTINUOUS OPTIMIZATION
FOR DIFFERENT INITIAL VALUE

iteration	p_1	p_2	p_3	error
1	0.005	0.25	0.005	2.214
10	0.0038	0.428	0.0066	1.050
20	0.0039	0.557	0.0075	0.756
30	0.0042	0.641	0.0079	0.643
40	0.0042	0.702	0.0083	0.566

TABLE XII

RESULTS OF CONTINUOUS OPTIMIZATION
FOR DIFFERENT INITIAL VALUE

iteration	p_1	p_2	p_3	error
1	0.015	0.1	0.005	0.0575
2	0.0157	0.095	0.0052	0.0533

TABLE XIII

RESULTS OF CONTINUOUS OPTIMIZATION
FOR DIFFERENT INITIAL VALUE

iteration	p_1	p_2	p_3	error
1	0.015	0.4	0.002	6.991
10	0.0174	0.267	0.0023	0.507
20	0.0178	0.221	0.0023	0.223
30	0.0180	0.194	0.0023	0.148
40	0.0181	0.175	0.0024	0.107
50	0.0182	0.160	0.0024	0.085

APPENDIX F

LISTING OF THE COMPUTER PROGRAM FOR THE SIMULATION OF THE EXAMPLE SYSTEM WITHOUT TRANSMISSION LINES

```
      program simulation1
c This program simulates the example system without
c transmission lines
      dimension x(14),y(4),p(4),xp(101),yp(101)
      data x/14*0./
      data u,ps,v/2.,500.,25./
      data p/4*1./
      data n,m,tf/14,4,1./
      write(6,103)
 103  format(3x,'Enter K ;Position Feedback Gain.')
      read(5,*) p(1)
      write(6,104)
 104  format(3x,'Enter Dm (cu in/rad.);Motor Displacement.')
      read(5,*) p(2)
      write(6,105)
 105  format(3x,'Enter K1 (cu in/rad/ma);Valve Flow Gain
     + Coefficient.')
      read(5,*) p(3)
      write(6,106)
 106  format(3x,'Enter Kp ;Pressure Feedback Gain.')
      read(5,*) p(4)
      write(6,100)
 100  format(3x,'Enter Time Step for Integration (sec).')
      read(5,*) delt
      ja=0
      ifin=ifix(tf/delt)
      xnu=float(ifin)/100.
      do 10 i=1,ifin
      t=float(i)*delt
      call runku(x,y,p,n,m,delt,1,x,y,t,u,ps,v)
      jb=ifix(i/xnu)
      j=jb-ja
      if(j.ne.1) go to 10
      xp(jb)=t
      yp(jb)=x(3)
      ja=jb
  10  continue
      call qckplt(xp,yp,100,'time$','position$',
     + 'Response$',0,5,0)
      stop
      end
c SUBROUTINE FOR 4TH ORDER RUNGE-KUTTA
      subroutine runku(x,y,p,n,m,delt,ind,sx,sy,t,u,ps,v)
      dimension x(14),dx(14),wk(56),p(4),sx(14),sy(14),
     +          y(4),dy(4),wj(16)
      selt=delt
      i1=2*n
      i2=3*n
      j1=2*m
      j2=3*m
      call confun(x,dx,y,p,u,ps,v)
      if(selt.lt.delt) then
      delt=selt
      return
```

```
      endif
 121  do 110 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+n)=dx(i)
 110  continue
      call confun(wk,dx,wj,p,u,ps,v)
      if(selt.lt.delt) then
      delt=selt
      return
      endif
 122  do 120 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+i1)=dx(i)
 120  continue
      call confun(wk,dx,wj,p,u,ps,v)
      if(selt.lt.delt) then
      delt=selt
      return
      endif
 123  do 130 i=1,n
       wk(i)=x(i)+delt*dx(i)
       wk(i+i2)=dx(i)
 130  continue
      call confun(wk,dx,wj,p,u,ps,v)
      if(selt.lt.delt) then
      delt=selt
      return
      endif
 124  do 140 i=1,n
      x(i)=x(i)+(wk(i+n)+2.*wk(i+i1)+2.*wk(i+i2)+dx(i))*delt/6.
 140  continue
      return
      end
c SUBROUTINE FOR STATE AND ALGEBRAIC EQUATIONS
      subroutine confun(x,dx,y,p,u,ps,v)
      dimension x(14),dx(14),y(4),p(4)
      b=165000.
      cs=4.4e-9
      xj=.00173
      cf=.17
      cd=1.12e5
      xmu=8e-6
      xk=p(1)
      dm=p(2)
      xk1=p(3)
      xk2=-.000954
      xkp=p(4)
      xi=u-xk*x(3)-xkp*x(1)
      y(1)=xk1*xi+0.5*xk2*x(1)
      dx(1)=2.*b/v*(y(1)-dm*x(2)-cs*dm*x(1)/xmu)
      dx(2)=(dm*x(1)-cd*xmu*dm*x(2)-cf*dm*x(1))/xj
      dx(3)=x(2)
      return
      end
```

APPENDIX G

LISTING OF THE COMPUTER PROGRAM FOR THE SIMULATION OF THE EXAMPLE SYSTEM WITH TRANSMISSION LINES

```
      program simulation2
c This program simulates the example system with transmission
c lines
      dimension x(14),y(4),p(4),xp(101),yp(101)
      data x/14*0./
      data u,ps,v/2.,500.,2.5/
      data p/4*0./
      data n,m,tf/14,4,1./
      write(6,103)
 103  format(3x,'Enter r0 (in);Inner Line Radius.')
      read(5,*) p(1)
      write(6,104)
 104  format(3x,'Enter Dm (cu in/rad);Motor Displacement.')
      read(5,*) p(2)
      write(6,105)
 105  format(3x,'Enter K1 (cu in/rad/ma);Valve Flow Gain
     + Coefficient.')
      read(5,*) p(3)
      write(6,106)
 106  format(3x,'Enter Kp ;Pressure Feedback Gain.')
      read(5,*) p(4)
      write(6,100)
 100  format(3x,'Enter Time Step for Integration (sec).')
      read(5,*) delt
      write(6,107)
 107  format(3x,'Enter L (in);Line Length.')
      read(5,*) xl
      ja=0
      ifin=ifix(tf/delt)
      xnu=float(ifin)/100.
      do 10 i=1,ifin
      t=float(i)*delt
      call runku(x,y,p,n,m,delt,1,x,y,t,u,ps,xl,v)
      jb=ifix(i/xnu)
      j=jb-ja
      if(j.ne.1) go to 10
      xp(jb)=t
      yp(jb)=x(14)
      ja=jb
  10  continue
      call qckplt(xp,yp,100,'time$','position$',
     + 'Response$',0,5,0)
      stop
      end
c SUBROUTINE FOR 4TH ORDER RUNGE-KUTTA
      subroutine runku(x,y,p,n,m,delt,ind,sx,sy,t,u,ps,xl,v)
      dimension x(14),dx(14),wk(56),p(4),sx(14),sy(14),
     +          y(4),dy(4),wj(16)
      selt=delt
      i1=2*n
      i2=3*n
      j1=2*m
      j2=3*m
      call confun(x,dx,y,p,u,ps,xl,v)
```

```
      if(selt.lt.delt) then
      delt=selt
      return
      endif
  121 do 110 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+n)=dx(i)
  110 continue
      call confun(wk,dx,wj,p,u,ps,xl,v)
      if(selt.lt.delt) then
      delt=selt
      return
      endif
  122 do 120 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+i1)=dx(i)
  120 continue
      call confun(wk,dx,wj,p,u,ps,xl,v)
      if(selt.lt.delt) then
      delt=selt
      return
      endif
  123 do 130 i=1,n
       wk(i)=x(i)+delt*dx(i)
       wk(i+i2)=dx(i)
  130 continue
      call confun(wk,dx,wj,p,u,ps,xl,v)
      if(selt.lt.delt) then
      delt=selt
      return
      endif
  124 do 140 i=1,n
      x(i)=x(i)+(wk(i+n)+2.*wk(i+i1)+2.*wk(i+i2)+dx(i))*delt/6.
  140 continue
      return
      end
c SUBROUTINE FOR STATE AND ALGEBRAIC EQUATIONS
      subroutine confun(x,dx,y,p,u,ps,xl,v)
      dimension x(14),dx(14),y(4),p(4)
      c1=.7138
      c2=.5273
      c3=1.3978
      c4=-.8403
      c0=50000.
      xn=.1
      te=xl/c0
      cl=xn*xl/c0/p(1)/p(1)
      z=c1*cl**c2
      cw=c4*cl
      w=c0*c3*exp(cl)/xl
      zs=81.5
      zt=zs*te
      yt=te/zs
      w2=w*w
```

```
zw=2.*z*w
b=165000.
cs=4.4e-9
xj=.00173
cf=.17
cd=1.12e5
xmu=8e-6
xk=2.
xi=u
dm=p(2)
xk1=p(3)
xk2=-.000954
xkp=p(4)
y(4)=u-xk*x(14)-xkp*x(12)
y(1)=xk1*y(4)+xk2*x(2)
y(2)=x(5)
y(3)=x(10)+x(12)
dx(1)=x(2)
dx(2)=-2.*w2*x(1)-zw*x(2)+w2*x(3)+w2*zt*y(1)
dx(3)=x(2)+y(3)
dx(4)=x(5)
dx(5)=-2.*w2*x(4)-zw*x(5)+w2*x(6)-w2*yt*y(3)
dx(6)=x(5)+y(1)
dx(7)=x(8)
tem1=w2*ys*te/xk2-2.*z*w
tem=-cd*xmu*dm*x(13)+dm*(1.-cf)*x(12)
tem2=xk*xk1/xk2/xj*tem
tem3=zw*xk*xk1/xk2*x(13)
dx(8)=-w2*x(7)+tem1*x(8)+w2*y(2)-tem2-tem3-w2*xk1*y(4)
dx(9)=x(10)
dx(10)=-2.*w2*x(9)-zw*x(10)+w2*x(11)+w2*zt*y(2)
dx(11)=x(10)+x(7)
dx(12)=2.*b/v*(-dm*x(13)-cs*dm/xmu*x(12)+x(5))
dx(13)=tem/xj
dx(14)=x(13)
return
end
```

APPENDIX H

LISTING OF THE COMPUTER PROGRAM FOR THE OPTIMIZATION OF THE EXAMPLE SYSTEM WITHOUT TRANSMISSION LINES

```
      program optimization1
c * This program optimizes following parameters of the position
c * control system without transmission lines.
c *    p(1)= Dm ; motor displacement
c *    p(2)= K1 ; valve coefficient
      dimension x(3),y(1),xa(3),xb(1),grad(4),p(4),pt(4),gx(4),
     +   po(4),pr(4),w(4),a(3),pf(3),pbran(21,4),ebran(21),
     +   branb(2),branc(2),pmin(4)
      common /c1/kind
      common /c2/anp
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      character*1 answ1,anp(4)
      data n,m,np/3,1,2/
      data u,ps/2.,500./
      data delt/0.02/
      data w/4*1./
      data p/.01512,.25,.01,.01/
      b=165000.
      cs=4.4e-9
      cf=.17
      cd=1.12e5
      xmu=8e-6
      xk=2.
      xk2=-.000954
      v=25.
      xj=.00173
      aa=1.
      xl=10.
      ibranch=2
      kind=1
      open (unit=7,file='wo1.d')
    1 write(6,117)
  117 format(/3x,'Enter the response you desire as a second
     + order.',/3x,'Enter natural frequency.')
      read(5,*) omega
      write(6,118)
  118 format(3x,'Enter damping coefficient.')
      read(5,*) zeta
      write(6,107)
  107 format(/3x,'Enter y for each parameter to be optimized.',
     +   /3x,'1. p(1)= (motor displacement) ;Optimize?  (y/n)')
      read(5,104) anp(1)
      write(6,108)
  108 format(/3x,'2. p(2)= (valve coefficient)  ;Optimize?
     + (y/n)')
      read(5,104) anp(2)
      write(6,102)
  102 format(/3x,'Enter initial guesses for the parameters'
     +       /3x,'you have chosen.')
      do 2 ia=1,np
      write(6,112) ia
  112 format(3x,'Initial guess for p(',i1,')?')
```

```
      read(5,*) p(ia)
    2 continue
      write(6,105)
  105 format(/3x,'Enter time step for the integration.')
      read(5,*) delt
      write(6,106)
  106 format(/3x,'Enter stopping criterion of the absolute
     + error.')
      read(5,*) stcr
      call optm(p,delt,n,m,grad,w,perf,u,ps,stcr,ibranch,np,xl)
c *** Checking initial guess
      if (kind.eq.2) then
      go to 1
      endif
  104 format(a1)
      stop
       end
c SUBROUTINE FOR OPTIMIZATION
      subroutine optm(p,delt,n,m,grad,w,perf,u,ps,stcr,
     + ibranch,np,xl)
      dimension p(4),grad(4),w(4),pt(4),po(4),pr(4),
     +          pf(3),a(3),gx(4)
      common /c1/kind
      common /c2/anp
      common /c3/aa,omega,zeta,z
      common /c4/t
      character*1 anp(4)
      do 30 i=1,np
   30 pr(i)=p(i)
      mi=1
      call calc(p,delt,n,m,grad,w,perf,u,ps,xl)
      preperf=perf
      do 10 j=1,np
      w(j)=abs(0.010*p(j)/grad(j))
   10 continue
      do 12 jj=1,np
      grad(jj)=grad(jj)*w(jj)
   12 continue
      write(6,109)
  109 format(///4x,'n',3x,'p(1)',6x,'p(2)',6x,'p(3)',6x,'p(4)',
     +  9x,'error')
      write(6,107) mi,p(1),p(2),perf
      write(7,107) mi,p(1),p(2),perf
  107 format(2x,i3,3x,2f12.6,3x,f12.6)
      do 11 mi=2,ibranch
       do 20 k1=1,3
       go to (21,22,23), k1
   21  do 29 i=1,np
   29  po(i)=p(i)
       a(1)=1.
       do 13 ia=1,np
   13  p(ia)=p(ia)+grad(ia)
       call perform(p,delt,n,m,perf,u,ps,xl)
       pf(1)=perf
```

```
      aind=3.5
      a(2)=aind
      a(3)=aind*2.
      do 14 i=1,np
      p(i)=po(i)+grad(i)*a(2)
 14   pt(i)=p(i)+grad(i)*a(2)
      do 16 i=1,np
 16   gx(i)=grad(i)
      go to 20
 22   call perform(p,delt,n,m,perf,u,ps,xl)
      pf(2)=perf
      do 17 i=1,np
 17   p(i)=pt(i)
      go to 20
 23   call perform(p,delt,n,m,perf,u,ps,xl)
      pf(3)=perf
 20   continue
      call quad(a,pf,alpha)
      do 15 i=1,np
      p(i)=po(i)
 15   grad(i)=gx(i)
      call project(p,np,alpha,grad,kind)
      if(kind.eq.2) go to 200
      do 25 ia=1,np
      if(anp(ia).ne.'y') then
      grad(ia)=0.
      endif
 25   continue
      do 18 i=1,np
 18   p(i)=p(i)+grad(i)*alpha
      call calc(p,delt,n,m,grad,w,perf,u,ps,xl)
       write(6,101) mi,p(1),p(2),perf
       write(7,101) mi,p(1),p(2),perf
 101   format(2x,i3,3x,2f12.6,3x,f12.6)
      gxt=0.
      do 19 i=1,np
 19   gxt=gxt+gx(i)**2
      if(gxt.lt.1.e-14) go to 200
 24   continue
       if(perf.lt.stcr) go to 200
       totpx=0.
       do 27 i=1,np
 27    totpx=totpx+((p(i)-pr(i))/p(i))**2
       if(totpx.lt.1.e-9) go to 200
       errdf=preperf-perf
       if(errdf.lt.1.e-5) go to 200
       preperf=perf
 11     continue
200     return
       end
c SUBROUTINE FOR PROJECTION OF PARAMETERS
      subroutine project(p,np,aind,grad,kind)
      dimension p(4),grad(4),psv(4)
      kind=1
```

```
10    do 11 i=1,np
      psv(i)=p(i)
      p(i)=p(i)+grad(i)*aind
11    continue
         inx=0
      if(p(1).lt..0001) then
         inx=1
      else if(p(1).gt.0.5) then
         inx=1
      endif
      if(p(2).lt.0.0001) then
         inx=2
      else if(p(2).gt.1.) then
         inx=2
      endif
      if(p(3).lt..0001) then
         inx=3
      else if(p(3).gt.1.) then
         inx=3
      endif
      if(p(4).lt.1.e-6) then
         inx=4
      else if(p(4).gt.5000.) then
         inx=4
      endif
       if(inx.gt.0) then
       aind=aind*0.75
       if(aind.lt.1.e-5) then
       go to (31,32,33,34),inx
 31    write(6,41)
 41    format(2x,'Try different guess for p(1) or reduce time
     + step.')
       go to 51
 32    write(6,42)
 42    format(2x,'Try different guess for p(2) or reduce time
     + step.')
       go to 51
 33    write(6,43)
 43    format(2x,'Try different guess for p(3) or reduce time
     + step.')
       go to 51
 34    write(6,44)
 44    format(2x,'Try different guess for p(4) or reduce time
     + step.')
 51    kind=2
       return
       endif
       do 20 j=1,np
       p(j)=psv(j)
 20    continue
       go to 10
       endif
       do 21 jj=1,np
       p(jj)=psv(jj)
```

```
  21    continue
        return
        end
c SUBROUTINE FOR QUADRATIC INTERPOLATION
        subroutine quad(a,pf,point)
        dimension a(3),pf(3)
        if(pf(3).lt.pf(1).and.pf(3).lt.pf(2)) then
         point=a(3)
         return
         end if
         coef11=1./(a(1)-a(2))/(a(1)-a(3))
         coef12=-(a(2)+a(3))*coef11
         coef21=1./(a(2)-a(1))/(a(2)-a(3))
         coef22=-(a(1)+a(3))*coef21
         coef31=1./(a(3)-a(1))/(a(3)-a(2))
         coef32=-(a(1)+a(2))*coef31
         c1=coef11*pf(1)+coef21*pf(2)+coef31*pf(3)
         c2=coef12*pf(1)+coef22*pf(2)+coef32*pf(3)
         if(c1.lt.0.) then
         point=1.
         return
         end if
         if(c1.eq.0.) go to 10
         point=-c2/c1*0.5
         go to 20
  10     if(c2.lt.0) then
         point=a(3)
         else
         point=a(1)
         end if
  20     if(point.lt.0.) then
         point=1.
         end if
         return
         end
c SUBROUTINE FOR 4TH ORDER RUNGE-KUTTA
        subroutine runku(x,y,p,n,m,delt,ind,sx,sy,t,u,ps,xl)
        dimension x(3),dx(3),wk(12),p(4),sx(3),sy(1),
       +          y(1),dy(1),wj(4)
        selt=delt
        i1=2*n
        i2=3*n
        j1=2*m
        j2=3*m
        go to (111,112), ind
 111    call confun(x,dx,y,p,u,ps,xl)
        call confuy(y,dy,x,p,u,ps,selt,xl)
        go to 121
 112    call adfun1(x,dx,y,p,sx,sy,t,u,ps,xl)
        call adfun2(y,dy,x,p,sx,sy,t,u,ps,xl)
 121    do 110 i=1,n
         wk(i)=x(i)+delt*dx(i)/2.
         wk(i+n)=dx(i)
 110    continue
```

```
      do 150 j=1,m
       wj(j)=y(j)+delt*dy(j)/2.
       wj(j+m)=dy(j)
 150  continue
      go to (115,116), ind
 115  call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 122
 116  call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
 122  do 120 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+i1)=dx(i)
 120  continue
      do 151 j=1,m
       wj(j)=y(j)+delt*dy(j)/2.
       wj(j+j1)=dy(j)
 151  continue
      go to (131,132), ind
 131  call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 123
 132  call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
 123  do 130 i=1,n
       wk(i)=x(i)+delt*dx(i)
       wk(i+i2)=dx(i)
 130  continue
      do 152 j=1,m
       wj(j)=y(j)+delt*dy(j)
       wj(j+j2)=dy(j)
 152  continue
      go to (135,136), ind
 135  call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 124
 136  call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
 124  do 140 i=1,n
      x(i)=x(i)+(wk(i+n)+2.*wk(i+i1)+2.*wk(i+i2)+dx(i))*delt/6.
 140  continue
      do 153 j=1,m
      y(j)=y(j)+(wj(j+m)+2.*wj(j+j1)+2.*wj(j+j2)+dy(j))*delt/6.
 153  continue
      return
      end
c SUBROUTINE FOR CALCULATION OF ERROR
      subroutine perform(p,delt,n,m,perf,u,ps,xl)
      dimension x(3),y(1),p(4)
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      external runku
      dm=p(1)
```

```
      xk1=p(2)
 1    cdelt=delt
      x(1)=0.
      x(2)=0.
      x(3)=0.
      y(1)=xk1*(u-xk*x(3))+.5*xk2*x(1)
      time=1.
      ifn=ifix(time/delt)+1
        do 20 i=1,ifn
        t=float(i-1)*delt
        if(i.eq.1) go to 21
        call runku(x,y,p,n,m,cdelt,1,x,y,t,u,ps,xl)
        if(cdelt.lt.delt) then
        delt=cdelt
        go to 1
        end if
 21     call desired(aa,omega,zeta,t,z)
        pz=x(3)-z
        pf=abs(pz)
        if(i.eq.ifn) go to 22
        if(i.gt.1) go to 23
        perf=pf
        go to 20
 23     itemp=i/2-(i-1)/2
        perf=perf+pf*(2.*itemp+2.)
        go to 20
 22     perf=(perf+pf)*delt/3.
 20     continue
        return
        end
c SUBROUTINE FOR FORWARD, BACKWARD INTEGRATIONS & GRADIENTS
      subroutine calc(p,delt,n,m,grad,w,perf,u,ps,xl)
      dimension
      x(3),xa(3),xb(1),xr(3,100001),sx(3),p(4),grad(4),
     +    gp(4),w(4),y(1),yr(1,100001),sy(1)
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      external runku,gradnt
      np=2.
 1    cdelt=delt
      preperf=perf
      time=1.
      ifn=ifix(time/delt)+1
      dm=p(1)
      xk1=p(2)
      x(1)=0.
      x(2)=0.
      x(3)=0.
      y(1)=xk1*(u-xk*x(3))+.5*xk2*x(1)
      do 25 mm=1,n
      xr(mm,1)=x(mm)
 25   continue
      do 45 mi=1,m
```

```
      yr(m1,1)=y(m1)
45    continue
        do 20 ix=1,ifn
        t=float(ix-1)*delt
        if(ix.eq.1) go to 21
        call runku(x,y,p,n,m,cdelt,1,x,y,t,u,ps,xl)
        if(cdelt.lt.delt) then
        delt=cdelt
        go to 1
        end if
        do 26 j=1,n
        xr(j,ix)=x(j)
26      continue
        do 46 jm=1,m
        yr(jm,ix)=y(jm)
46      continue
21      call desired(aa,omega,zeta,t,z)
        pz=x(3)-z
        pf=abs(pz)
        if(ix.eq.ifn) go to 22
        if(ix.gt.1) go to 23
        perf=pf
        go to 20
23      itemp=ix/2-(ix-1)/2
        perf=perf+pf*(2.*itemp+2.)
        go to 20
22      perf=(perf+pf)*delt/3.
20      continue
       xa(1)=0.
       xa(2)=0.
       xa(3)=0.
       xb(1)=0.
       dm=-delt
        do 30 i=1,ifn
        t=float(ifn-i)*delt
        do 27 k=1,n
        sx(k)=xr(k,ifn+1-i)
 27     continue
        do 47 km=1,m
        sy(km)=yr(km,ifn+1-i)
 47     continue
        if(i.eq.1) go to 31
        dm=-delt
        call runku(xa,xb,p,n,m,dm,2,sx,sy,t,u,ps,xl)
31      call gradnt(xa,xb,sx,sy,p,u,ps,gp,xl)
        if(i.eq.ifn) go to 32
         if(i.gt.1) go to 33
         do 41 m1=1,np
         grad(m1)=gp(m1)
41       continue
         go to 30
33        itemp=i/2-(i-1)/2
         do 42 m2=1,np
          grad(m2)=grad(m2)+gp(m2)*(2.*itemp+2.)
```

```
 42      continue
         go to 30
 32       do 43 m3=1,np
           grad(m3)=(grad(m3)+gp(m3))*delt/3.
 43       continue
 30       continue
         do 44 m4=1,np
         grad(m4)=grad(m4)*w(m4)
 44      continue
         return
         end
c SUBROUTINE FOR STATE EQUATIONS
      subroutine confun(x,dx,y,p,u,ps,xl)
      dimension x(3),dx(3),y(1),p(4)
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      dx(1)=2.*b/v*(y(1)-dm*x(2)-cs*dm/xmu*x(1))
      dx(2)=((1.-cf)*dm*x(1)-cd*xmu*dm*x(2))/xj
      dx(3)=x(2)
      return
      end
c SUBROUTINE FOR ALGEBRAIC EQUATIONS
      subroutine confuy(y,dy,x,p,u,ps,delt,xl)
      dimension y(1),dy(1),x(3),p(4)
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      tem1=y(1)-dm*x(2)-cs*dm/xmu*x(1)
      dy(1)=-xk*xk1*x(2)+xk2*b/v*tem1
      return
      end
c SUBROUTINE FOR ADJOINT STATE EQUATIONS
      subroutine adfun1(xa,dxa,xb,p,x,y,t,u,ps,xl)
      dimension xa(3),dxa(3),xb(1),p(4),x(3),y(1)
      common /c3/aa,omega,zeta,z
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      call desired(aa,omega,zeta,t,dsy)
      tem1=cs*dm/xmu
      tem2=b/v*tem1
      tem4=(1.-cf)*dm/xj
      dxa(1)=xa(1)*tem2*2.-xa(2)*tem4+xb(1)*xk2*tem2
      dxa(2)=xa(1)*b/v*dm*2.+xa(2)*cd*xmu*dm/xj-xa(3)-xb(1)*
     + (-xk*xk1-xk2*b/v*dm)
      if(x(3).ge.dsy) then
      dxa(3)=1.
      else
      dxa(3)=-1.
      endif
      return
      end
c SUBROUTINE FOR ADJOINT ALGEBRAIC EQUATIONS
```

```
      subroutine adfun2(xb,dxb,xa,p,x,y,t,u,ps,xl)
      dimension xb(1),dxb(1),xa(3),p(4),x(3),y(1)
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      dxb(1)=-xa(1)*b/v*2.-xb(1)*xk2*b/v
      return
      end
c SUBROUTINE FOR GRADIENT VECTOR
      subroutine gradnt(xa,xb,x,y,p,u,ps,gp,xl)
      dimension xa(3),xb(1),x(3),y(1),p(4),gp(4)
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      tem=x(2)+cs/xmu*x(1)
      tem1=tem*b/v
      tem2=cs*dm/xmu
      tem3=xk2+tem2
      tem4=((1.-cf)*x(1)-cd*xmu*x(2))/xj
      gp(1)=-xa(1)*tem1*2.+xa(2)*tem4-xb(1)*b/v*xk2*tem
      gp(2)=-xb(1)*xk*x(2)
      return
      end
c SUBROUTINE FOR DESIRED RESPONSE
c   omega: natural frequency
c   zeta:  damping coefficient
c   omd:   damped natural frequency
c   aa:    initial condition
c   z:     response
      subroutine desired(aa,omega,zeta,t,z)
      r1=sqrt(1./zeta/zeta-1.)
      phi=atan(r1)
      omd=omega*sqrt(1.-zeta*zeta)
      z1=-zeta*omega*t
      z2=exp(z1)
      z3=z2/sin(phi)
      z4=omd*t+phi
      z5=sin(z4)
      z=aa*(1.-z3*z5)
      return
      end
```

APPENDIX I

LISTING OF THE COMPUTER PROGRAM FOR THE OPTIMIZATION OF THE EXAMPLE SYSTEM WITHOUT TRANSMISSION LINES (WITH PRESSURE FEEDBACK)

```
      program optimization2
c * This program optimizes following parameters of the positon
c * control system without transmission lines through use of
c * pressure feedback.
c *    p(1)= Dm ; motor displacement
c *    p(2)= K1 ; valve coefficient
c *    p(3)= Kp ; pressure feedback gain
      dimension x(3),y(1),xa(3),xb(1),grad(4),p(4),pt(4),gx(4),
     +   po(4),pr(4),w(4),a(3),pf(3),pbran(21,4),ebran(21),
     +   branb(2),branc(2),pmin(4)
      common /c1/kind
      common /c2/anp
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      character*1 answ1,anp(4)
      data n,m,np/3,1,3/
      data u,ps/2.,500./
      data delt/0.02/
      data w/4*1./
      data p/.01512,.25,.01,.01/
      b=165000.
      cs=4.4e-9
      cf=.17
      cd=1.12e5
      xmu=8e-6
      xk=2.
      xk2=-.000954
      v=25.
      xj=.00173
      aa=1.
      xl=10.
      ibranch=2
      kind=1
      open (unit=7,file='wo1.d')
    1 write(6,117)
  117 format(/3x,'Enter the response you desire as a second
     + order.', /3x,'Enter natural frequency.')
      read(5,*) omega
      write(6,118)
  118 format(3x,'Enter damping coefficient.')
      read(5,*) zeta
      write(6,107)
  107 format(/3x,'Enter y for each parameter to be optimized.',
     +   /3x,'1. p(1)= (motor displacement) ;Optimize?  (y/n)')
      read(5,104) anp(1)
      write(6,108)
  108 format(/3x,'2. p(2)= (valve coefficient)  ;Optimize?
     + (y/n)')
      read(5,104) anp(2)
      write(6,109)
  109 format(/3x,'3. p(3)= (pressure feedback
     + gain);Optimize?(y/n)')
      read(5,104) anp(3)
```

```
      write(6,102)
  102  format(/3x,'Enter initial guesses for the parameters'
     +       /3x,'you have chosen.')
      do 2 ia=1,np
      write(6,112) ia
  112  format(3x,'Initial guess for p(',i1,')?')
      read(5,*) p(ia)
    2  continue
      write(6,105)
  105  format(/3x,'Enter time step for the integration.')
      read(5,*) delt
      write(6,106)
  106  format(/3x,'Enter stopping criterion of the absolute
     + error.')
      read(5,*) stcr
      ibranch=75
      call optm(p,delt,n,m,grad,w,perf,u,ps,stcr,ibranch,np,xl)
c *** Checking initial guess
      if (kind.eq.2) then
      go to 1
      endif
  104   format(a1)
       stop
       end
c SUBROUTINE FOR OPTIMIZATION
      subroutine optm(p,delt,n,m,grad,w,perf,u,ps,stcr,
     + ibranch,np,xl)
      dimension p(4),grad(4),w(4),pt(4),po(4),pr(4),
     +          pf(3),a(3),gx(4)
      common /c1/kind
      common /c2/anp
      common /c3/aa,omega,zeta,z
      common /c4/t
      character*1 anp(4)
      do 30 i=1,np
   30  pr(i)=p(i)
      mi=1
      call calc(p,delt,n,m,grad,w,perf,u,ps,xl)
      preperf=perf
      do 10 j=1,np
      w(j)=abs(0.010*p(j)/grad(j))
   10  continue
      do 12 jj=1,np
      grad(jj)=grad(jj)*w(jj)
   12  continue
      write(6,109)
  109  format(///4x,'n',3x,'p(1)',6x,'p(2)',6x,'p(3)',6x,'p(4)',
     +  9x,'error')
      write(6,107) mi,p(1),p(2),p(3),perf
      write(7,107) mi,p(1),p(2),p(3),perf
  107  format(2x,i3,3x,3f12.6,3x,f12.6)
      do 11 mi=2,ibranch
       do 20 k1=1,3
       go to (21,22,23), k1
```

```
21    do 29 i=1,np
29    po(i)=p(i)
      a(1)=1.
      do 13 ia=1,np
13    p(ia)=p(ia)+grad(ia)
      call perform(p,delt,n,m,perf,u,ps,xl)
      pf(1)=perf
      aind=3.5
      a(2)=aind
      a(3)=aind*2.
      do 14 i=1,np
      p(i)=po(i)+grad(i)*a(2)
14    pt(i)=p(i)+grad(i)*a(2)
      do 16 i=1,np
16    gx(i)=grad(i)
      go to 20
22    call perform(p,delt,n,m,perf,u,ps,xl)
      pf(2)=perf
      do 17 i=1,np
17    p(i)=pt(i)
      go to 20
23    call perform(p,delt,n,m,perf,u,ps,xl)
      pf(3)=perf
20    continue
      call quad(a,pf,alpha)
      do 15 i=1,np
      p(i)=po(i)
15    grad(i)=gx(i)
      call project(p,np,alpha,grad,kind)
      if(kind.eq.2) go to 200
      do 25 ia=1,np
      if(anp(ia).ne.'y') then
      grad(ia)=0.
      endif
25    continue
      do 18 i=1,np
18    p(i)=p(i)+grad(i)*alpha
      call calc(p,delt,n,m,grad,w,perf,u,ps,xl)
       write(6,101) mi,p(1),p(2),p(3),perf
       write(7,101) mi,p(1),p(2),p(3),perf
101    format(2x,i3,3x,3f12.6,3x,f12.6)
      gxt=0.
      do 19 i=1,np
19    gxt=gxt+gx(i)**2
      if(gxt.lt.1.e-14) go to 200
24    continue
       if(perf.lt.stcr) go to 200
       totpx=0.
       do 27 i=1,np
27     totpx=totpx+((p(i)-pr(i))/p(i))**2
       if(totpx.lt.1.e-9) go to 200
       errdf=preperf-perf
       if(errdf.lt.1.e-5) go to 200
       preperf=perf
```

```
 11     continue
200     return
       end
c SUBROUTINE FOR PROJECTION OF PARAMETERS
      subroutine project(p,np,aind,grad,kind)
      dimension p(4),grad(4),psv(4)
      kind=1
 10   do 11 i=1,np
      psv(i)=p(i)
      p(i)=p(i)+grad(i)*aind
 11   continue
         inx=0
      if(p(1).lt..0001) then
         inx=1
      else if(p(1).gt.0.5) then
         inx=1
      endif
      if(p(2).lt.0.0001) then
         inx=2
      else if(p(2).gt.1.) then
         inx=2
      endif
      if(p(3).lt..00001) then
         inx=3
      else if(p(3).gt.1.) then
         inx=3
      endif
      if(p(4).lt.1.e-6) then
         inx=4
      else if(p(4).gt.5000.) then
         inx=4
      endif
       if(inx.gt.0) then
       aind=aind*0.75
       if(aind.lt.1.e-5) then
       go to (31,32,33,34),inx
 31    write(6,41)
 41    format(2x,'Try different guess for p(1) or reduce time
     + step.')
       go to 51
 32    write(6,42)
 42    format(2x,'Try different guess for p(2) or reduce time
     + step.')
       go to 51
 33    write(6,43)
 43    format(2x,'Try different guess for p(3) or reduce time
     + step.')
       go to 51
 34    write(6,44)
 44    format(2x,'Try different guess for p(4) or reduce time
     + step.')
 51    kind=2
       return
       endif
```

```
      do 20 j=1,np
      p(j)=psv(j)
 20   continue
      go to 10
      endif
      do 21 jj=1,np
      p(jj)=psv(jj)
 21   continue
     return
     end
c SUBROUTINE FOR QUADRATIC INTERPOLATION
     subroutine quad(a,pf,point)
     dimension a(3),pf(3)
     if(pf(3).lt.pf(1).and.pf(3).lt.pf(2)) then
      point=a(3)
      return
      end if
      coef11=1./(a(1)-a(2))/(a(1)-a(3))
      coef12=-(a(2)+a(3))*coef11
      coef21=1./(a(2)-a(1))/(a(2)-a(3))
      coef22=-(a(1)+a(3))*coef21
      coef31=1./(a(3)-a(1))/(a(3)-a(2))
      coef32=-(a(1)+a(2))*coef31
      c1=coef11*pf(1)+coef21*pf(2)+coef31*pf(3)
      c2=coef12*pf(1)+coef22*pf(2)+coef32*pf(3)
      if(c1.lt.0.) then
      point=1.
      return
      end if
      if(c1.eq.0.) go to 10
      point=-c2/c1*0.5
      go to 20
 10   if(c2.lt.0) then
      point=a(3)
      else
      point=a(1)
      end if
 20   if(point.lt.0.) then
      point=1.
      end if
      return
      end
c SUBROUTINE FOR 4TH ORDER RUNGE-KUTTA
     subroutine runku(x,y,p,n,m,delt,ind,sx,sy,t,u,ps,xl)
     dimension x(3),dx(3),wk(12),p(4),sx(3),sy(1),
    +          y(1),dy(1),wj(4)
     selt=delt
     i1=2*n
     i2=3*n
     j1=2*m
     j2=3*m
     go to (111,112), ind
 111 call confun(x,dx,y,p,u,ps,xl)
     call confuy(y,dy,x,p,u,ps,selt,xl)
```

```
      go to 121
  112 call adfun1(x,dx,y,p,sx,sy,t,u,ps,xl)
      call adfun2(y,dy,x,p,sx,sy,t,u,ps,xl)
  121 do 110 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+n)=dx(i)
  110 continue
      do 150 j=1,m
       wj(j)=y(j)+delt*dy(j)/2.
       wj(j+m)=dy(j)
  150 continue
      go to (115,116), ind
  115 call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 122
  116 call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
  122 do 120 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+i1)=dx(i)
  120 continue
      do 151 j=1,m
       wj(j)=y(j)+delt*dy(j)/2.
       wj(j+j1)=dy(j)
  151 continue
      go to (131,132), ind
  131 call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 123
  132 call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
  123 do 130 i=1,n
       wk(i)=x(i)+delt*dx(i)
       wk(i+i2)=dx(i)
  130 continue
      do 152 j=1,m
       wj(j)=y(j)+delt*dy(j)
       wj(j+j2)=dy(j)
  152 continue
      go to (135,136), ind
  135 call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 124
  136 call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
  124 do 140 i=1,n
      x(i)=x(i)+(wk(i+n)+2.*wk(i+i1)+2.*wk(i+i2)+dx(i))*delt/6.
  140 continue
      do 153 j=1,m
      y(j)=y(j)+(wj(j+m)+2.*wj(j+j1)+2.*wj(j+j2)+dy(j))*delt/6.
  153 continue
      return
      end
c SUBROUTINE FOR CALCULATION OF ERROR
```

```
      subroutine perform(p,delt,n,m,perf,u,ps,xl)
      dimension x(3),y(1),p(4)
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      external runku
      dm=p(1)
      xk1=p(2)
      xkp=p(3)
 1    cdelt=delt
      x(1)=0.
      x(2)=0.
      x(3)=0.
      y(1)=xk1*(u-xk*x(3)-xkp*x(1))+.5*xk2*x(1)
      time=1.
      ifn=ifix(time/delt)+1
        do 20 i=1,ifn
        t=float(i-1)*delt
        if(i.eq.1) go to 21
        call runku(x,y,p,n,m,cdelt,1,x,y,t,u,ps,xl)
        if(cdelt.lt.delt) then
        delt=cdelt
        go to 1
        end if
 21     call desired(aa,omega,zeta,t,z)
        pz=x(3)-z
c        pf=pz*pz
        pf=abs(pz)
        if(i.eq.ifn) go to 22
        if(i.gt.1) go to 23
        perf=pf
        go to 20
 23     itemp=i/2-(i-1)/2
        perf=perf+pf*(2.*itemp+2.)
        go to 20
 22     perf=(perf+pf)*delt/3.
 20     continue
        return
        end
c SUBROUTINE FOR FORWARD, BACKWARD INTEGRATIONS & GRADIENTS
      subroutine calc(p,delt,n,m,grad,w,perf,u,ps,xl)
      dimension
      x(3),xa(3),xb(1),xr(3,100001),sx(3),p(4),grad(4),
     1    gp(4),w(4),y(1),yr(1,100001),sy(1)
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      external runku,gradnt
      np=3.
 1    cdelt=delt
      preperf=perf
      time=1.
      ifn=ifix(time/delt)+1
      dm=p(1)
```

```
      xk1=p(2)
      xkp=p(3)
      x(1)=0.
      x(2)=0.
      x(3)=0.
      y(1)=xk1*(u-xk*x(3)-xkp*x(1))+.5*xk2*x(1)
      do 25 mm=1,n
      xr(mm,1)=x(mm)
   25 continue
      do 45 mi=1,m
      yr(mi,1)=y(mi)
   45 continue
        do 20 ix=1,ifn
        t=float(ix-1)*delt
        if(ix.eq.1) go to 21
        call runku(x,y,p,n,m,cdelt,1,x,y,t,u,ps,xl)
        if(cdelt.lt.delt) then
        delt=cdelt
        go to 1
        end if
        do 26 j=1,n
        xr(j,ix)=x(j)
   26   continue
        do 46 jm=1,m
        yr(jm,ix)=y(jm)
   46   continue
   21   call desired(aa,omega,zeta,t,z)
        pz=x(3)-z
c        pf=pz*pz
        pf=abs(pz)
        if(ix.eq.ifn) go to 22
        if(ix.gt.1) go to 23
        perf=pf
        go to 20
   23   itemp=ix/2-(ix-1)/2
        perf=perf+pf*(2.*itemp+2.)
        go to 20
   22   perf=(perf+pf)*delt/3.
   20   continue
       xa(1)=0.
       xa(2)=0.
       xa(3)=0.
       xb(1)=0.
       dm=-delt
        do 30 i=1,ifn
        t=float(ifn-i)*delt
        do 27 k=1,n
        sx(k)=xr(k,ifn+1-i)
    27  continue
        do 47 km=1,m
        sy(km)=yr(km,ifn+1-i)
    47  continue
        if(i.eq.1) go to 31
        dm=-delt
```

```
      call runku(xa,xb,p,n,m,dm,2,sx,sy,t,u,ps,xl)
31    call gradnt(xa,xb,sx,sy,p,u,ps,gp,xl)
      if(i.eq.ifn) go to 32
       if(i.gt.1) go to 33
       do 41 m1=1,np
       grad(m1)=gp(m1)
41     continue
       go to 30
33      itemp=i/2-(i-1)/2
       do 42 m2=1,np
        grad(m2)=grad(m2)+gp(m2)*(2.*itemp+2.)
42     continue
       go to 30
32      do 43 m3=1,np
         grad(m3)=(grad(m3)+gp(m3))*delt/3.
43      continue
30      continue
       do 44 m4=1,np
       grad(m4)=grad(m4)*w(m4)
44     continue
       return
       end
c SUBROUTINE FOR STATE EQUATIONS
      subroutine confun(x,dx,y,p,u,ps,xl)
      dimension x(3),dx(3),y(1),p(4)
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      xkp=p(3)
      dx(1)=2.*b/v*(y(1)-dm*x(2)-cs*dm/xmu*x(1))
      dx(2)=((1.-cf)*dm*x(1)-cd*xmu*dm*x(2))/xj
      dx(3)=x(2)
      return
      end
c SUBROUTINE FOR ALGEBRAIC EQUATIONS
      subroutine confuy(y,dy,x,p,u,ps,delt,xl)
      dimension y(1),dy(1),x(3),p(4)
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      xkp=p(3)
      tem1=y(1)-dm*x(2)-cs*dm/xmu*x(1)
      tem2=xk2-2.*xk1*xkp
      dy(1)=-xk*xk1*x(2)+tem2*b/v*tem1
      return
      end
c SUBROUTINE FOR ADJOINT STATE EQUATIONS
      subroutine adfun1(xa,dxa,xb,p,x,y,t,u,ps,xl)
      dimension xa(3),dxa(3),xb(1),p(4),x(3),y(1)
      common /c3/aa,omega,zeta,z
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      xkp=p(3)
```

```
      call desired(aa,omega,zeta,t,dsy)
      tem1=cs*dm/xmu
      tem2=b/v*tem1
      tem3=xk2-2.*xk1*xkp
      tem4=(1.-cf)*dm/xj
      dxa(1)=xa(1)*tem2*2.-xa(2)*tem4-xb(1)*tem3*b/v*tem1
      dxa(2)=xa(1)*b/v*dm*2.+xa(2)*cd*xmu*dm/xj-xa(3)-xb(1)*
     + (xk*xk1+tem3*b/v*dm)
      if(x(3).ge.dsy) then
      dxa(3)=1.
      else
      dxa(3)=-1.
      endif
      return
      end
c SUBROUTINE FOR ADJOINT ALGEBRAIC EQUATIONS
      subroutine adfun2(xb,dxb,xa,p,x,y,t,u,ps,xl)
      dimension xb(1),dxb(1),xa(3),p(4),x(3),y(1)
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      xkp=p(3)
      dxb(1)=-xa(1)*b/v*2.+xb(1)*2.*xk1*xkp*b/v-
     +  xb(1)*xk2*b/v
      return
      end
c SUBROUTINE FOR GRADIENT VECTOR
      subroutine gradnt(xa,xb,x,y,p,u,ps,gp,xl)
      dimension xa(3),xb(1),x(3),y(1),p(4),gp(4)
      common /p1/b,cs,cf,cd,xmu,xk,xk2,v,xj
      dm=p(1)
      xk1=p(2)
      xkp=p(3)
      tem=x(2)+cs/xmu*x(1)
      tem1=tem*b/v
      tem2=cs*dm/xmu
      tem3=xk2+tem2
      tem4=((1.-cf)*x(1)-cd*xmu*x(2))/xj
      te=y(1)-dm*x(2)-tem2*x(1)
      tem5=xk2-2.*xk1*xkp
      gp(1)=-xa(1)*tem1*2.+xa(2)*tem4+xb(1)*tem5*tem1
      gp(2)=-xb(1)*xk*x(2)-xb(1)*xkp*b/v*te*2.
      gp(3)=-xb(1)*xk1*b/v*te*2.
      return
      end
c SUBROUTINE FOR DESIRED RESPONSE
c   omega: natural frequency
c   zeta:  damping coefficient
c   omd:   damped natural frequency
c   aa:    initial condition
c   z:     response
       subroutine desired(aa,omega,zeta,t,z)
       r1=sqrt(1./zeta/zeta-1.)
       phi=atan(r1)
```

```
omd=omega*sqrt(1.-zeta*zeta)
z1=-zeta*omega*t
z2=exp(z1)
z3=z2/sin(phi)
z4=omd*t+phi
z5=sin(z4)
z=aa*(1.-z3*z5)
return
end
```

APPENDIX J

LISTING OF THE COMPUTER PROGRAM FOR THE OPTIMIZATION OF THE EXAMPLE SYSTEM WITH TRANSMISSION LINES

```
      program optimization3
c * This program optimizes following parameters of the position
c * control system with transmission lines.
c *    p(1)= r  ; radius of the line
c *    p(2)= Dm ; motor displacement
c *    p(3)= K1 ; valve flow gain coefficient
      dimension x(14),y(4),xa(14),xb(4),grad(4),p(4),pt(4),
     + gx(4),po(4),pr(4),w(4),a(3),pf(3),pbran(21,4),ebran(21),
     +   branb(2),branc(2),pmin(4)
      common /c1/kind
      common /c2/anp
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      character*1 answ1,anp(4)
      data n,m,np/14,4,3/
      data u,ps/2.,500./
      data delt/0.0002/
      data w/4*1./
      data p/.15,.01512,.25,.001/
      b=165000.
      v=2.5
      xj=.00173
      xmu=8e-6
      cd=1.12e5
      cf=.17
      cs=4.4e-9
      xk=2.
      xk2=-.000954
      aa=1.
      ibranch=2
      kind=1
      open (unit=7,file='noprsfb.d')
    1 write(6,117)
  117 format(/3x,'Enter the response you desire as a second
     +  order.'  /3x,'Enter natural frequency.')
      read(5,*) omega
      write(6,118)
  118 format(3x,'Enter damping coefficient.')
      read(5,*) zeta
      write(6,107)
  107 format(/3x,'Enter y for each parameter to be optimized.',
     +  /3x,'1. p(1)= (radius of line)     ;Optimize?  (y/n)')
      read(5,104) anp(1)
      write(6,108)
  108 format(/3x,'2. p(2)= (motor diceplacement);Optimize?
     + (y/n)')
      read(5,104) anp(2)
      write(6,109)
  109 format(/3x,'3. p(3)= (valve coefficient)  ;Optimize?
     + (y/n)')
      read(5,104) anp(3)
      write(6,102)
  102 format(/3x,'Enter initial guesses for the parameters'
```

```
     +       /3x,'you have chosen.')
      do 2 ia=1,np
      write(6,112) ia
 112  format(3x,'Initial guess for p(',i1,')?')
      read(5,*) p(ia)
   2  continue
      write(6,121)
 121  format(/3x,'Enter line length in inches.')
      read(5,*) xl
      write(6,106)
 106  format(/3x,'Enter stopping criterion of the absolute
     + error.')
      read(5,*) stcr
      ibranch=75
      call optm(p,delt,n,m,grad,w,perf,u,ps,stcr,ibranch,np,xl)
c *** Checking initial guess
      if (kind.eq.2) then
      go to 1
      endif
 104  format(a1)
      stop
       end
c SUBROUTINE FOR OPTIMIZATION
      subroutine optm(p,delt,n,m,grad,w,perf,u,ps,stcr,ibranch,
     +                np,xl)
      dimension p(4),grad(4),w(4),pt(4),po(4),pr(4),
     +          pf(3),a(3),gx(4)
      common /c1/kind
      common /c2/anp
      common /c3/aa,omega,zeta,z
      common /c4/t
      character*1 anp(4)
      do 30 i=1,np
  30  pr(i)=p(i)
      mi=1
      call calc(p,delt,n,m,grad,w,perf,u,ps,xl)
      preperf=perf
      do 10 j=1,np
      w(j)=abs(0.010*p(j)/grad(j))
  10  continue
      do 12 jj=1,np
      grad(jj)=grad(jj)*w(jj)
  12  continue
      write(6,109)
 109  format(///4x,'n',3x,'p(1)',6x,'p(2)',6x,'p(3)',6x,'p(4)',
     +  9x,'error')
      write(6,107) mi,p(1),p(2),p(3),p(4),perf
      write(7,107) mi,p(1),p(2),p(3),p(4),perf
 107  format(2x,i3,3x,4f12.6,3x,f12.6)
      do 11 mi=2,ibranch
       do 20 kl=1,3
       go to (21,22,23), kl
  21   do 29 i=1,np
  29   po(i)=p(i)
```

```
      a(1)=1.
      do 13 ia=1,np
 13   p(ia)=p(ia)+grad(ia)
      call perform(p,delt,n,m,perf,u,ps,xl)
      pf(1)=perf
      aind=3.5
      a(2)=aind
      a(3)=aind*2.
      do 14 i=1,np
      p(i)=po(i)+grad(i)*a(2)
 14   pt(i)=p(i)+grad(i)*a(2)
      do 16 i=1,np
 16   gx(i)=grad(i)
      go to 20
 22   call perform(p,delt,n,m,perf,u,ps,xl)
      pf(2)=perf
      do 17 i=1,np
 17   p(i)=pt(i)
      go to 20
 23   call perform(p,delt,n,m,perf,u,ps,xl)
      pf(3)=perf
 20   continue
      call quad(a,pf,alpha)
      do 15 i=1,np
      p(i)=po(i)
 15   grad(i)=gx(i)
      call project(p,np,alpha,grad,kind)
      if(kind.eq.2) go to 200
      do 25 ia=1,np
      if(anp(ia).ne.'y') then
      grad(ia)=0.
      endif
 25   continue
      do 18 i=1,np
 18   p(i)=p(i)+grad(i)*alpha
      call calc(p,delt,n,m,grad,w,perf,u,ps,xl)
       write(6,101) mi,p(1),p(2),p(3),p(4),perf
       write(7,101) mi,p(1),p(2),p(3),p(4),perf
 101   format(2x,i3,3x,4f12.6,3x,f12.6)
      gxt=0.
      do 19 i=1,np
 19   gxt=gxt+gx(i)**2
      if(gxt.lt.1.e-14) go to 200
 24   continue
       if(perf.lt.stcr) go to 200
       totpx=0.
       do 27 i=1,np
 27    totpx=totpx+((p(i)-pr(i))/p(i))**2
       if(totpx.lt.1.e-9) go to 200
       errdf=preperf-perf
       if(errdf.lt.1.e-5) go to 200
       preperf=perf
 11     continue
200     return
```

```
        end
c SUBROUTINE FOR PROJECTION OF PARAMETERS
      subroutine project(p,np,aind,grad,kind)
      dimension p(4),grad(4),psv(4)
      common /c2/anp
      character*1 anp(4)
      kind=1
 10   do 11 i=1,np
      psv(i)=p(i)
      p(i)=p(i)+grad(i)*aind
 11   continue
         inx=0
      if(p(1).lt..001) then
         inx=1
      else if(p(1).gt.10.) then
         inx=1
      endif
      if(p(2).lt.0.0001) then
         inx=2
      else if(p(2).gt.0.5) then
         inx=2
      endif
      if(p(3).lt..0001) then
         inx=3
      else if(p(3).gt.1.) then
         inx=3
      endif
 12    if(inx.gt.0) then
       aind=aind*0.75
       if(aind.lt.1.e-5) then
       go to (31,32,33,34),inx
 31    write(6,41)
 41    format(2x,'Try different guess for p(1) or reduce time
     + step.')
       go to 51
 32    write(6,42)
 42    format(2x,'Try different guess for p(2) or reduce time
     + step.')
       go to 51
 33    write(6,43)
 43    format(2x,'Try different guess for p(3) or reduce time
     + step.')
       go to 51
 34    write(6,44)
 44    format(2x,'Try different guess for p(4) or reduce time
     + step.')
 51    kind=2
       return
       endif
       do 20 j=1,np
       p(j)=psv(j)
 20    continue
       go to 10
       endif
```

```
      do 21 jj=1,np
      p(jj)=psv(jj)
   21 continue
      return
      end
c SUBROUTINE FOR QUADRATIC INTERPOLATION
      subroutine quad(a,pf,point)
      dimension a(3),pf(3)
      if(pf(3).lt.pf(1).and.pf(3).lt.pf(2)) then
       point=a(3)
       return
       end if
       coef11=1./(a(1)-a(2))/(a(1)-a(3))
       coef12=-(a(2)+a(3))*coef11
       coef21=1./(a(2)-a(1))/(a(2)-a(3))
       coef22=-(a(1)+a(3))*coef21
       coef31=1./(a(3)-a(1))/(a(3)-a(2))
       coef32=-(a(1)+a(2))*coef31
       c1=coef11*pf(1)+coef21*pf(2)+coef31*pf(3)
       c2=coef12*pf(1)+coef22*pf(2)+coef32*pf(3)
       if(c1.lt.0.) then
       point=1.
       return
       end if
       if(c1.eq.0.) go to 10
       point=-c2/c1*0.5
       go to 20
   10  if(c2.lt.0) then
       point=a(3)
       else
       point=a(1)
       end if
   20  if(point.lt.0.) then
       point=1.
       end if
       return
       end
c SUBROUTINE FOR 4TH ORDER RUNGE-KUTTA
      subroutine runku(x,y,p,n,m,delt,ind,sx,sy,t,u,ps,xl)
      dimension x(14),dx(14),wk(56),p(4),sx(14),sy(4),
     +          y(4),dy(4),wj(16)
      selt=delt
      i1=2*n
      i2=3*n
      j1=2*m
      j2=3*m
      go to (111,112), ind
  111 call confun(x,dx,y,p,u,ps,xl)
      call confuy(y,dy,x,p,u,ps,selt,xl)
      go to 121
  112 call adfun1(x,dx,y,p,sx,sy,t,u,ps,xl)
      call adfun2(y,dy,x,p,sx,sy,t,u,ps,xl)
  121 do 110 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
```

```
      wk(i+n)=dx(i)
  110 continue
      do 150 j=1,m
       wj(j)=y(j)+delt*dy(j)/2.
       wj(j+m)=dy(j)
  150 continue
      go to (115,116), ind
  115 call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 122
  116 call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
  122 do 120 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+i1)=dx(i)
  120 continue
      do 151 j=1,m
       wj(j)=y(j)+delt*dy(j)/2.
       wj(j+j1)=dy(j)
  151 continue
      go to (131,132), ind
  131 call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 123
  132 call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
  123 do 130 i=1,n
       wk(i)=x(i)+delt*dx(i)
       wk(i+i2)=dx(i)
  130 continue
      do 152 j=1,m
       wj(j)=y(j)+delt*dy(j)
       wj(j+j2)=dy(j)
  152 continue
      go to (135,136), ind
  135 call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 124
  136 call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
  124 do 140 i=1,n
      x(i)=x(i)+(wk(i+n)+2.*wk(i+i1)+2.*wk(i+i2)+dx(i))*delt/6.
  140 continue
      do 153 j=1,m
      y(j)=y(j)+(wj(j+m)+2.*wj(j+j1)+2.*wj(j+j2)+dy(j))*delt/6.
  153 continue
      return
      end
c SUBROUTINE FOR CALCULATION OF ERROR
      subroutine perform(p,delt,n,m,perf,u,ps,xl)
      dimension x(14),y(4),p(4)
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
```

```
      external runku
      xk1=p(3)
 1    cdelt=delt
      x(1)=0.
      x(2)=0.
      x(3)=0.
      x(4)=0.
      x(5)=0.
      x(6)=0.
      x(7)=0.
      x(8)=0.
      x(9)=0.
      x(10)=0.
      x(11)=0.
      x(12)=0.
      x(13)=0.
      x(14)=0.
      y(4)=u-xk*x(14)
      y(1)=xk1*y(4)+xk2*x(2)
      y(2)=x(5)
      y(3)=x(10)+x(12)
      time=1.
      ifn=ifix(time/delt)+1
        do 20 i=1,ifn
        t=float(i-1)*delt
        if(i.eq.1) go to 21
        call runku(x,y,p,n,m,cdelt,1,x,y,t,u,ps,xl)
        if(cdelt.lt.delt) then
        delt=cdelt
        go to 1
        end if
 21     call desired(aa,omega,zeta,t,z)
        pz=x(14)-z
c        pf=pz*pz
        pf=abs(pz)
        if(i.eq.ifn) go to 22
        if(i.gt.1) go to 23
        perf=pf
        go to 20
 23     itemp=i/2-(i-1)/2
        perf=perf+pf*(2.*itemp+2.)
        go to 20
 22     perf=(perf+pf)*delt/3.
 20     continue
        return
        end
c SUBROUTINE FOR FORWARD, BACKWARD INTEGRATIONS & GRADIENTS
      subroutine calc(p,delt,n,m,grad,w,perf,u,ps,xl)
      dimension x(14),xa(14),xb(4),xr(14,100001),sx(14),p(4),
     + grad(4),gp(4),w(4),y(4),yr(4,100001),sy(4)
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      external runku,gradnt
```

```
      np=3
    1 cdelt=delt
      preperf=perf
      time=1.
      ifn=ifix(time/delt)+1
      xk1=p(3)
      x(1)=0.
      x(2)=0.
      x(3)=0.
      x(4)=0.
      x(5)=0.
      x(6)=0.
      x(7)=0.
      x(8)=0.
      x(9)=0.
      x(10)=0.
      x(11)=0.
      x(12)=0.
      x(13)=0.
      x(14)=0.
      y(4)=u-xk*x(14)
      y(1)=xk1*y(4)+xk2*x(2)
      y(2)=x(5)
      y(3)=x(10)+x(12)
      do 25 mm=1,n
      xr(mm,1)=x(mm)
   25 continue
      do 45 mi=1,m
      yr(mi,1)=y(mi)
   45 continue
        do 20 ix=1,ifn
        t=float(ix-1)*delt
        if(ix.eq.1) go to 21
        call runku(x,y,p,n,m,cdelt,1,x,y,t,u,ps,xl)
        if(cdelt.lt.delt) then
        delt=cdelt
        go to 1
        end if
        do 26 j=1,n
        xr(j,ix)=x(j)
   26   continue
        do 46 jm=1,m
        yr(jm,ix)=y(jm)
   46   continue
   21   call desired(aa,omega,zeta,t,z)
        pz=x(14)-z
c        pf=pz*pz
        pf=abs(pz)
        if(ix.eq.ifn) go to 22
        if(ix.gt.1) go to 23
        perf=pf
        go to 20
   23   itemp=ix/2-(ix-1)/2
        perf=perf+pf*(2.*itemp+2.)
```

```
      go to 20
22    perf=(perf+pf)*delt/3.
20    continue
     xa(1)=0.
     xa(2)=0.
     xa(3)=0.
     xa(4)=0.
     xa(5)=0.
     xa(6)=0.
     xa(7)=0.
     xa(8)=0.
     xa(9)=0.
     xa(10)=0.
     xa(11)=0.
     xa(12)=0.
     xa(13)=0.
     xa(14)=0.
     xb(1)=0.
     xb(2)=0.
     xb(3)=0.
     xb(4)=0.
     dm=-delt
      do 30 i=1,ifn
      t=float(ifn-i)*delt
      do 27 k=1,n
      sx(k)=xr(k,ifn+1-i)
 27   continue
      do 47 km=1,m
      sy(km)=yr(km,ifn+1-i)
 47   continue
      if(i.eq.1) go to 31
      dm=-delt
      call runku(xa,xb,p,n,m,dm,2,sx,sy,t,u,ps,xl)
31    call gradnt(xa,xb,sx,sy,p,u,ps,gp,xl)
      if(i.eq.ifn) go to 32
       if(i.gt.1) go to 33
       do 41 m1=1,np
       grad(m1)=gp(m1)
41     continue
       go to 30
33      itemp=i/2-(i-1)/2
       do 42 m2=1,np
        grad(m2)=grad(m2)+gp(m2)*(2.*itemp+2.)
42     continue
       go to 30
32      do 43 m3=1,np
         grad(m3)=(grad(m3)+gp(m3))*delt/3.
43      continue
30      continue
       do 44 m4=1,np
       grad(m4)=grad(m4)*w(m4)
44     continue
       return
       end
```

```
c SUBROUTINE FOR STATE EQUATIONS
      subroutine confun(x,dx,y,p,u,ps,xl)
      dimension x(14),dx(14),y(4),p(4)
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      common /p2/rho,c1,c2,c3,c4
      rho=8e-5
      xk1=p(3)
      radius=p(1)
      area=3.141592*radius*radius
      c0=sqrt(b/rho)
      te=xl/c0
      zs=sqrt(rho*b)/area
      ys=1./zs
      xnu=xmu/rho
      pe=xnu*xl/c0/p(1)/p(1)
c --- Curve-fit of omega & zeta of line
      c1=.7138
      c2=.5273
      c3=1.3978
      c4=-.8403
      pe2=c4*pe
      z=c1*pe**c2
      w=c0*c3*exp(pe2)/xl
      w2=w*w
      dx(1)=x(2)
      dx(2)=-2.*w2*x(1)-2.*z*w*x(2)+w2*x(3)+w2*zs*te*y(1)
      dx(3)=x(2)+y(3)
      dx(4)=x(5)
      dx(5)=-2.*w2*x(4)-2.*z*w*x(5)+w2*x(6)-w2*ys*te*y(3)
      dx(6)=x(5)+y(1)
      dx(7)=x(8)
      tem1=w2*ys*te/xk2-2.*z*w
      tem2=-cd*xmu*p(2)*x(13)+p(2)*(1.-cf)*x(12)
      dx(8)=-w2*x(7)+tem1*x(8)+w2*y(2)-xk*xk1/xk2/xj*tem2-
     +  2.*z*w*xk*xk1/xk2*x(13)-w2*xk1*y(4)
      dx(9)=x(10)
      dx(10)=-2.*w2*x(9)-2.*z*w*x(10)+w2*x(11)+w2*zs*te*y(2)
      dx(11)=x(10)+x(7)
      dx(12)=2.*b/v*(-p(2)*x(13)-cs*p(2)/xmu*x(12)+x(5))
      dx(13)=1./xj*tem2
      dx(14)=x(13)
      return
      end
c SUBROUTINE FOR ALGEBRAIC EQUATIONS
      subroutine confuy(y,dy,x,p,u,ps,delt,xl)
      dimension y(4),dy(4),x(14),p(4)
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      common /p2/rho,c1,c2,c3,c4
      xk1=p(3)
      radius=p(1)
      area=3.141592*radius*radius
      c0=sqrt(b/rho)
      te=xl/c0
      zs=sqrt(rho*b)/area
```

```
      ys=1./zs
      xnu=xmu/rho
      pe=xnu*xl/c0/p(1)/p(1)
      pe2=c4*pe
      z=c1*pe**c2
      w=c0*c3*exp(pe2)/xl
      w2=w*w
      tem1=-2.*w2*x(1)-2.*z*w*x(2)+w2*x(3)+w2*zs*te*y(1)
      tem2=-2.*w2*x(4)-2.*z*w*x(5)+w2*x(6)-w2*ys*te*y(3)
      tem3=-2.*w2*x(9)-2.*z*w*x(10)+w2*x(11)+w2*zs*te*y(2)
      tem4=-p(2)*x(13)-cs*p(2)/xmu*x(12)+x(5)
      dy(1)=-xk*xk1*x(13)+xk2*tem1
      dy(2)=tem2
      dy(3)=tem3+2.*b/v*tem4
      dy(4)=-xk*x(13)
      return
      end
c SUBROUTINE FOR ADJOINT STATE EQUATIONS
      subroutine adfun1(xa,dxa,xb,p,x,y,t,u,ps,xl)
      dimension xa(14),dxa(14),xb(4),p(4),x(14),y(4)
      common /c3/aa,omega,zeta,z
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      common /p2/rho,c1,c2,c3,c4
      call desired(aa,omega,zeta,t,dsy)
      xk1=p(3)
      radius=p(1)
      area=3.141592*radius*radius
      c0=sqrt(b/rho)
      te=xl/c0
      zs=sqrt(rho*b)/area
      ys=1./zs
      xnu=xmu/rho
      pe=xnu*xl/c0/p(1)/p(1)
      pe2=c4*pe
      z=c1*pe**c2
      w=c0*c3*exp(pe2)/xl
      w2=w*w
      dxa(1)=2.*w2*xa(2)+2.*xk2*w2*xb(1)
      dxa(2)=-xa(1)+2.*z*w*xa(2)-xa(3)+2.*xk2*z*w*xb(1)
      dxa(3)=-w2*xa(2)-xk2*w2*xb(1)
      dxa(4)=2.*w2*xa(5)+2.*w2*xb(2)
      dxa(5)=-xa(4)+2.*z*w*xa(5)-xa(6)-2.*b/v*xa(12)+2.*z*w
     +  *xb(2)-2.*b/v*xb(3)
      dxa(6)=-w2*xa(5)-w2*xb(2)
      dxa(7)=w2*xa(8)-xa(11)
      dxa(8)=-xa(7)+(-w2*ys*te/xk2+2.*z*w)*xa(8)
      dxa(9)=2.*w2*xa(10)+2.*w2*xb(3)
      dxa(10)=-xa(9)+2.*z*w*xa(10)-xa(11)+2.*z*w*xb(3)
      dxa(11)=-w2*xa(10)-w2*xb(3)
      tem1=p(2)*(1.-cf)/xj
      tem2=cs*p(2)/xmu
      tem3=b/v*tem2
      tem4=p(2)*tem1-tem2*tem2*b/v
      dxa(12)=xk*xk1/xk2*tem1*xa(8)+2.*tem3*xa(12)-tem1*xa(13)
```

```
     +   +2.*tem3*xb(3)
         dxa(13)=xk*xk1/xk2*(-cd*xmu*p(2)/xj+2.*z*w)*xa(8)+2.*
     +   b*p(2)/v*xa(12)+cd*xmu*p(2)/xj*xa(13)-xa(14)+
     + xk*xk1*xb(1)+2.*b*p(2)/v*xb(3)+xk*xb(4)
         if(x(14).ge.dsy) then
         dxa(14)=1.
         else
         dxa(14)=-1.
         endif
         return
         end
c SUBROUTINE FOR ADJOINT ALGEBRAIC EQUATIONS
         subroutine adfun2(xb,dxb,xa,p,x,y,t,u,ps,xl)
         dimension xb(4),dxb(4),xa(14),p(4),x(14),y(4)
         common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
         common /p2/rho,c1,c2,c3,c4
         xk1=p(3)
         radius=p(1)
         area=3.141592*radius*radius
         c0=sqrt(b/rho)
         te=xl/c0
         zs=sqrt(rho*b)/area
         ys=1./zs
         xnu=xmu/rho
         pe=xnu*xl/c0/p(1)/p(1)
         pe2=c4*pe
         z=c1*pe**c2
         w=c0*c3*exp(pe2)/xl
         w2=w*w
         dxb(1)=-w2*zs*te*xa(2)-xa(6)-xk2*w2*zs*te*xb(1)
         dxb(2)=-w2*xa(8)-w2*zs*te*xa(10)-w2*zs*te*xb(3)
         dxb(3)=-xa(3)+w2*ys*te*xa(5)+w2*ys*te*xb(2)
         dxb(4)=w2*xk1*xa(8)
         return
         end
c SUBROUTINE FOR GRADIENT VECTOR
         subroutine gradnt(xa,xb,x,y,p,u,ps,gp,xl)
         dimension xa(14),xb(4),x(14),y(4),p(4),gp(4)
         common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
         common /p2/rho,c1,c2,c3,c4
         xk1=p(3)
         radius=p(1)
         area=3.141592*radius*radius
         c0=sqrt(b/rho)
         te=xl/c0
         zs=sqrt(rho*b)/area
         ys=1./zs
         xnu=xmu/rho
         c1=xnu*xl/c0
         pe=c1/p(1)/p(1)
         pe2=c4*pe
         z=c1*pe**c2
         w=c0*c3*exp(pe2)/xl
         w2=w*w
```

```
      c2m=c2-1.
      dz=c1*c2*pe**(c2m)*c1*(-2.)/p(1)/p(1)/p(1)
      dw=-2.*c4*c1*w/p(1)/p(1)/p(1)
      dzw=dz*w+z*dw
      dwt=2.*w*dw*te
      w2w=2.*w*dw
      w4w=4.*w*dw
      z2=2.*dzw
      zsdw=zs*dwt
      ysdw=ys*dwt
      tem1=ysdw/xk2-z2
      tema=-w4w*x(1)-z2*x(2)+w2w*x(3)+zsdw*y(1)
      temb=-w4w*x(4)-z2*x(5)+w2w*x(6)-ysdw*y(3)
      temc=-w4w*x(9)-z2*x(10)+w2w*x(11)+zsdw*y(2)
      gp(1)=xa(2)*tema+xa(5)*temb+xa(8)*(-w2w*x(7)+tem1*x(8)+
     + w2w*y(2)-z2*xk*xk1/xk2*x(13)-w2w*xk1*y(4))+xa(10)*temc+
     + xb(1)*xk2*tema+xb(2)*temb+xb(3)*temc
      tem2=cs*b/xmu/v
      tem3=-b/v*x(13)-tem2*x(12)
      tem4=-cd*xmu*p(2)*x(13)+p(2)*(1.-cf)*x(12)
      tem5=2.*tem4/xj
      gp(2)=xa(8)*xk*xk1/xk2/xj*(cd*xmu*x(13)-(1.-cf)*x(12))+2.*
     + xa(12)*tem3+xa(13)*(-cd*xmu/xj*x(13)+(1.-cf)/xj*x(12))+
     + 2.*xb(3)*tem3
      gp(3)=xa(8)*(-xk/xk2/xj*tem4-2.*z*w*xk/xk2*x(13)-
     + w2*y(4))-xb(1)*xk*x(13)
      return
      end
c SUBROUTINE FOR DESIRED RESPONSE
      subroutine desired(aa,omega,zeta,t,z)
      r1=sqrt(1./zeta/zeta-1.)
      phi=atan(r1)
      omd=omega*sqrt(1.-zeta*zeta)
      z1=-zeta*omega*t
      z2=exp(z1)
      z3=z2/sin(phi)
      z4=omd*t+phi
      z5=sin(z4)
      z=aa*(1.-z3*z5)
      return
      end
```

APPENDIX K

LISTING OF THE COMPUTER PROGRAM FOR THE OPTIMIZATION OF THE EXAMPLE SYSTEM WITH TRANSMISSION LINES (WITH PRESSURE FEEDBACK)

```
      program optimization4
c * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
c * This program is a complete program to optimize the             *
c * position control system with transmission lines through       *
c * use of feedback and has 'branch and bound' procedure.         *
c * p(1)= r0 ; radius of the line                                  *
c * p(2)= Dm ; motor displacement                                  *
c * p(3)= K1 ; valve flow gain coefficient                         *
c * p(4)= Kp ; pressure feedback gain                              *
c *                                                               *
c *  Variable Definitions:                                        *
c *                                                               *
c *    np      ; number of parameters                             *
c *    n       ; number of state variables                        *
c *    m       ; number of algebraic variables                    *
c *    x       ; state variable                                   *
c *    y       ; algebraic variable                               *
c *    xa      ; adjoint state variable                           *
c *    xb      ; adjoint algebraic variable                       *
c *    grad    ; gradient                                         *
c *    pt,po   ; temporary value of parameter                     *
c *    pr      ; previous value of parameter                      *
c *    gx      ; temporary value of gradient                      *
c *    w       ; weight factor                                    *
c *    a       ; alpha (factor for quadratic interpolation)       *
c *    pf      ; performance or error                             *
c *    branb   ; branched parameter                               *
c *    pmin    ; minimum parameter                                *
c *    u       ; input or current to the valve                    *
c *    ps      ; supply pressure                                  *
c *    b       ; bulk modulus of fluid                            *
c *    v       ; volume under compression                         *
c *    xj      ; load inertia                                     *
c *    xmu     ; viscosity                                        *
c *    cd      ; drag coefficient of the motor                    *
c *    cf      ; friction coefficient of the motor                *
c *    cs      ; slip coefficient of the motor                    *
c *    xk      ; position feedback gain                           *
c *    xk2     ; slope of pressure flow characteristics of the    *
c *              valve                                            *
c *    xl      ; transmission line length                         *
c *    aa      ; initial condition                                *
c *    kind    ; index for checking initial guesses               *
c *    stcr    ; stopping criterion                               *
c *    delt    ; integration time step                            *
c * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      dimension x(14),y(4),xa(14),xb(4),grad(4),p(4),pt(4),
     +gx(4),po(4),pr(4),w(4),a(3),pf(3),pbran(21,4),ebran(21),
     + branb(2),branc(2),pmin(4)
      common /c1/kind
      common /c2/anp
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
```

```
      character*1 answ1,anp(4)
      data n,m,np/14,4,4/
      data u,ps/2.,500./
      data delt/0.0002/
      data w/4*1./
      data p/.15,.01512,.25,.001/
      b=165000.
      v=2.5
      xj=.00173
      xmu=8e-6
      cd=1.12e5
      cf=.17
      cs=4.4e-9
      xk=2.
      xk2=-.000954
      aa=1.
      ibranch=2
      kind=1
      open (unit=7,file='optm.d')
    1 write(6,117)
  117 format(/3x,'Enter the response you desire as a second
     + order.',   /3x,'Enter natural frequency.')
      read(5,*) omega
      write(6,118)
  118 format(3x,'Enter damping coefficient.')
      read(5,*) zeta
      write(6,120)
  120 format(3x,'Enter steady state value.')
      read(5,*) aa
      write(6,107)
  107 format(/3x,'Enter y for each parameter to be optimized.',
     +   /3x,'1. p(1)= (radius of line)     ;Optimize?  (y/n)')
      read(5,104) anp(1)
      write(6,108)
  108 format(/3x,'2. p(2)= (motor diceplacement);Optimize?
     + (y/n)')
      read(5,104) anp(2)
      write(6,109)
  109 format(/3x,'3. p(3)= (valve coefficient)  ;Optimize?
     + (y/n)')
      read(5,104) anp(3)
      write(6,110)
  110 format(/3x,'4. p(4)= (feedback gain)      ;Optimize?
     + (y/n)')
      read(5,104) anp(4)
      write(6,102)
  102 format(/3x,'Enter initial guesses for the parameters'
     +       /3x,'you have chosen.')
      do 2 ia=1,np
      write(6,112) ia
  112 format(3x,'Initial guess for p(',i1,')?')
      read(5,*) p(ia)
    2 continue
      write(6,121)
```

```
 121  format(/3x,'Enter line length in inches.')
      read(5,*) xl
      write(6,106)
 106  format(/3x,'Enter stopping criterion of the absolute
     + error.')
      read(5,*) stcr
      ibranch=75
      call optm(p,delt,n,m,grad,w,perf,u,ps,stcr,ibranch,np,xl)
c *** Checking initial guess
      if (kind.eq.2) then
      go to 1
      endif
       do 42 k=1,np
       pmin(k)=p(k)
  42   continue
       perfmin=perf
c *** Branching
      write(6,116)
 116  format(/2x,'Branch and Bound starts.')
       kb=1
       kbpr=kb
       do 49 lb=1,2
c* Save the previous parameters
       do 50 ib=1,np
   50  pr(ib)=p(ib)
c* Get four branches initially
       if(lb.ne.2) then
        call choose(pr,ichoose)
        else
        ichoose=2
       endif
       call branch(pr,branb,branc)
       if(ichoose.eq.1) then
        anp(2)='n'
        anp(3)='y'
        else
        anp(2)='y'
        anp(3)='n'
       endif
       p(1)=pr(1)
       p(4)=pr(4)
       do 51 jb=1,2
       if(ichoose.eq.1) then
        p(3)=pr(3)
          if(jb.eq.1) then
           p(2)=branb(1)
          else
           p(2)=branb(2)
          endif
       else
        p(2)=pr(2)
          if(jb.eq.1) then
           p(3)=branc(1)
          else
```

```
            p(3)=branc(2)
           endif
         endif
         kb=kb+1
         ibranch=5
         call optm(p,delt,n,m,grad,w,perf,u,ps,stcr,ibranch,np,xl)
         do 52 iib=1,np
         pbran(kb,iib)=p(iib)
   52    ebran(kb)=perf
         call check(p,perf,ichoose,n,m,np,u,ps)
   51    continue
         if(lb.eq.1) then
          go to 49
         else
          kbpr=kbpr+1
          do 53 j1=1,np
   53     p(j1)=pbran(kbpr,j1)
         endif
   49    continue
         chv=2.
         do 54 km=1,20
         if(ebran(km).le.chv) then
         perf=1.
         do 55 kj1=1,np
   55    p(kj1)=pbran(km,kj1)
         call choose(p,ichoose)
         call check(p,perf,ichoose,n,m,np,u,ps)
         endif
         chv=chv+1.
   54    continue
         write(6,103)
  103    format(/'Do you want to try another guess? (y/n)')
         read(5,104) answ1
  104    format(a1)
         if(answ1.eq.'y') then
         go to 1
         end if
         stop
         end
c SUBROUTINE FOR OPTIMIZATION
      subroutine optm(p,delt,n,m,grad,w,perf,u,ps,stcr,ibranch,
     +                np,xl)
      dimension p(4),grad(4),w(4),pt(4),po(4),pr(4),
     +          pf(3),a(3),gx(4)
      common /c1/kind
      common /c2/anp
      common /c3/aa,omega,zeta,z
      common /c4/t
      character*1 anp(4)
      do 30 i=1,np
   30 pr(i)=p(i)
      mi=1
      call calc(p,delt,n,m,grad,w,perf,u,ps,xl)
      preperf=perf
```

```
      do 10 j=1,np
      if (grad(j).eq.0.) then
      grad(j)=0.01*p(j)
      else
      w(j)=abs(0.010*p(j)/grad(j))
      grad(j)=grad(j)*w(j)
      endif
 10   continue
      write(6,109)
109   format(///4x,'n',3x,'p(1)',6x,'p(2)',6x,'p(3)',6x,'p(4)',
     +  9x,'error')
      write(6,107) mi,p(1),p(2),p(3),p(4),perf
      write(7,107) mi,p(1),p(2),p(3),p(4),perf
107   format(2x,i3,3x,4f12.6,3x,f12.6)
      do 11 mi=2,ibranch
       do 20 kl=1,3
       go to (21,22,23), kl
 21    do 29 i=1,np
 29    po(i)=p(i)
       a(1)=1.
       do 13 ia=1,np
 13    p(ia)=p(ia)+grad(ia)
       call perform(p,delt,n,m,perf,u,ps,xl)
       pf(1)=perf
       aind=3.5
       a(2)=aind
       a(3)=aind*2.
       do 14 i=1,np
       p(i)=po(i)+grad(i)*a(2)
 14    pt(i)=p(i)+grad(i)*a(2)
       do 16 i=1,np
 16    gx(i)=grad(i)
       go to 20
 22    call perform(p,delt,n,m,perf,u,ps,xl)
       pf(2)=perf
       do 17 i=1,np
 17    p(i)=pt(i)
       go to 20
 23    call perform(p,delt,n,m,perf,u,ps,xl)
       pf(3)=perf
 20    continue
       call quad(a,pf,alpha)
       do 15 i=1,np
       p(i)=po(i)
 15    grad(i)=gx(i)
       call project(p,np,alpha,grad,kind)
       if(kind.eq.2) go to 200
       do 25 ia=1,np
       if(anp(ia).ne.'y') then
       grad(ia)=0.
       endif
 25    continue
       do 18 i=1,np
 18    p(i)=p(i)+grad(i)*alpha
```

```
      call calc(p,delt,n,m,grad,w,perf,u,ps,xl)
       write(6,101) mi,p(1),p(2),p(3),p(4),perf
       write(7,101) mi,p(1),p(2),p(3),p(4),perf
 101   format(2x,i3,3x,4f12.6,3x,f12.6)
      gxt=0.
      do 19 i=1,np
 19   gxt=gxt+gx(i)**2
      if(gxt.lt.1.e-14) go to 200
 24   continue
       if(perf.lt.stcr) go to 200
       totpx=0.
       do 27 i=1,np
 27    totpx=totpx+((p(i)-pr(i))/p(i))**2
       if(totpx.lt.1.e-9) go to 200
       errdf=preperf-perf
       if(errdf.lt.1.e-5) go to 200
       preperf=perf
 11     continue
200     return
       end
c SUBROUTINE FOR PROJECTION OF PARAMETERS
      subroutine project(p,np,aind,grad,kind)
      dimension p(4),grad(4),psv(4)
      kind=1
 10   do 11 i=1,np
      psv(i)=p(i)
      p(i)=p(i)+grad(i)*aind
 11   continue
         inx=0
      if(p(1).lt..001) then
         inx=1
      else if(p(1).gt.10.) then
         inx=1
      endif
      if(p(2).lt.0.001) then
         inx=2
      else if(p(2).gt.1.) then
         inx=2
      endif
      if(p(3).lt..0001) then
         inx=3
      else if(p(3).gt.1.) then
         inx=3
      endif
      if(p(4).lt.1.e-6) then
         inx=4
      else if(p(4).gt.5000.) then
         inx=4
      endif
       if(inx.gt.0) then
       aind=aind*0.75
       if(aind.lt.1.e-5) then
       go to (31,32,33,34),inx
 31    write(6,41)
```

```
 41    format(2x,'Try different guess for p(1) or reduce time
     + step.')
       go to 51
 32    write(6,42)
 42    format(2x,'Try different guess for p(2) or reduce time
     + step.')
       go to 51
 33    write(6,43)
 43    format(2x,'Try different guess for p(3) or reduce time
     + step.')
       go to 51
 34    write(6,44)
 44    format(2x,'Try different guess for p(4) or reduce time
     + step.')
 51    kind=2
       return
       endif
       do 20 j=1,np
       p(j)=psv(j)
 20    continue
       go to 10
       endif
       do 21 jj=1,np
       p(jj)=psv(jj)
 21    continue
      return
      end
c SUBROUTINE FOR QUADRATIC INTERPOLATION
      subroutine quad(a,pf,point)
      dimension a(3),pf(3)
      if(pf(3).lt.pf(1).and.pf(3).lt.pf(2)) then
       point=a(3)
       return
       end if
       coef11=1./(a(1)-a(2))/(a(1)-a(3))
       coef12=-(a(2)+a(3))*coef11
       coef21=1./(a(2)-a(1))/(a(2)-a(3))
       coef22=-(a(1)+a(3))*coef21
       coef31=1./(a(3)-a(1))/(a(3)-a(2))
       coef32=-(a(1)+a(2))*coef31
       c1=coef11*pf(1)+coef21*pf(2)+coef31*pf(3)
       c2=coef12*pf(1)+coef22*pf(2)+coef32*pf(3)
       if(c1.lt.0.) then
       point=1.
       return
       end if
       if(c1.eq.0.) go to 10
       point=-c2/c1*0.5
       go to 20
 10    if(c2.lt.0) then
       point=a(3)
       else
       point=a(1)
       end if
```

```
 20    if(point.lt.0.) then
       point=1.
       end if
       return
       end
c SUBROUTINE FOR 4TH ORDER RUNGE-KUTTA
      subroutine runku(x,y,p,n,m,delt,ind,sx,sy,t,u,ps,xl)
      dimension x(14),dx(14),wk(56),p(4),sx(14),sy(4),
     +          y(4),dy(4),wj(16)
      selt=delt
      i1=2*n
      i2=3*n
      j1=2*m
      j2=3*m
      go to (111,112), ind
 111  call confun(x,dx,y,p,u,ps,xl)
      call confuy(y,dy,x,p,u,ps,selt,xl)
      go to 121
 112  call adfun1(x,dx,y,p,sx,sy,t,u,ps,xl)
      call adfun2(y,dy,x,p,sx,sy,t,u,ps,xl)
 121  do 110 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+n)=dx(i)
 110  continue
      do 150 j=1,m
       wj(j)=y(j)+delt*dy(j)/2.
       wj(j+m)=dy(j)
 150  continue
      go to (115,116), ind
 115  call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 122
 116  call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
 122  do 120 i=1,n
       wk(i)=x(i)+delt*dx(i)/2.
       wk(i+i1)=dx(i)
 120  continue
      do 151 j=1,m
       wj(j)=y(j)+delt*dy(j)/2.
       wj(j+j1)=dy(j)
 151  continue
      go to (131,132), ind
 131  call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 123
 132  call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
 123  do 130 i=1,n
       wk(i)=x(i)+delt*dx(i)
       wk(i+i2)=dx(i)
 130  continue
      do 152 j=1,m
       wj(j)=y(j)+delt*dy(j)
```

```
      wj(j+j2)=dy(j)
  152 continue
      go to (135,136), ind
  135 call confun(wk,dx,wj,p,u,ps,xl)
      call confuy(wj,dy,wk,p,u,ps,selt,xl)
      go to 124
  136 call adfun1(wk,dx,wj,p,sx,sy,t,u,ps,xl)
      call adfun2(wj,dy,wk,p,sx,sy,t,u,ps,xl)
  124 do 140 i=1,n
      x(i)=x(i)+(wk(i+n)+2.*wk(i+i1)+2.*wk(i+i2)+dx(i))*delt/6.
  140 continue
      do 153 j=1,m
      y(j)=y(j)+(wj(j+m)+2.*wj(j+j1)+2.*wj(j+j2)+dy(j))*delt/6.
  153 continue
      return
      end
c SUBROUTINE FOR CALCULATION OF ERROR
      subroutine perform(p,delt,n,m,perf,u,ps,xl)
      dimension x(14),y(4),p(4)
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      external runku
      xk1=p(3)
      xkp=p(4)
    1 cdelt=delt
      x(1)=0.
      x(2)=0.
      x(3)=0.
      x(4)=0.
      x(5)=0.
      x(6)=0.
      x(7)=0.
      x(8)=0.
      x(9)=0.
      x(10)=0.
      x(11)=0.
      x(12)=0.
      x(13)=0.
      x(14)=0.
      y(4)=u-xk*x(14)-xkp*x(12)
      y(1)=xk1*y(4)+xk2*x(2)
      y(2)=x(5)
      y(3)=x(10)+x(12)
      time=.75
      ifn=ifix(time/delt)+1
        do 20 i=1,ifn
        t=float(i-1)*delt
        if(i.eq.1) go to 21
        call runku(x,y,p,n,m,cdelt,1,x,y,t,u,ps,xl)
        if(cdelt.lt.delt) then
        delt=cdelt
        go to 1
        end if
```

```
 21     call desired(aa,omega,zeta,t,z)
        pz=x(14)-z
        pf=abs(pz)
        if(i.eq.ifn) go to 22
        if(i.gt.1) go to 23
        perf=pf
        go to 20
 23     itemp=i/2-(i-1)/2
        perf=perf+pf*(2.*itemp+2.)
        go to 20
 22     perf=(perf+pf)*delt/3.
 20     continue
        return
        end
C
C SUBROUTINE FOR FORWARD, BACKWARD INTEGRATIONS & GRADIENTS
      subroutine calc(p,delt,n,m,grad,w,perf,u,ps,xl)
      dimension x(14),xa(14),xb(4),xr(14,100001),sx(14),
     + p(4),grad(4),gp(4),w(4),y(4),yr(4,100001),sy(4)
      common /c3/aa,omega,zeta,z
      common /c4/t
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      external runku,gradnt
      np=4.
 1    cdelt=delt
      preperf=perf
      time=.75
      ifn=ifix(time/delt)+1
      xk1=p(3)
      xkp=p(4)
      x(1)=0.
      x(2)=0.
      x(3)=0.
      x(4)=0.
      x(5)=0.
      x(6)=0.
      x(7)=0.
      x(8)=0.
      x(9)=0.
      x(10)=0.
      x(11)=0.
      x(12)=0.
      x(13)=0.
      x(14)=0.
      y(4)=u-xk*x(14)-xkp*x(12)
      y(1)=xk1*y(4)+xk2*x(2)
      y(2)=x(5)
      y(3)=x(10)+x(12)
      do 25 mm=1,n
      xr(mm,1)=x(mm)
 25   continue
      do 45 mi=1,m
      yr(mi,1)=y(mi)
 45   continue
```

```
      do 20 ix=1,ifn
      t=float(ix-1)*delt
      if(ix.eq.1) go to 21
      call runku(x,y,p,n,m,cdelt,1,x,y,t,u,ps,xl)
      if(cdelt.lt.delt) then
      delt=cdelt
      go to 1
      end if
      do 26 j=1,n
      xr(j,ix)=x(j)
26    continue
      do 46 jm=1,m
      yr(jm,ix)=y(jm)
46    continue
21    call desired(aa,omega,zeta,t,z)
      pz=x(14)-z
      pf=abs(pz)
      if(ix.eq.ifn) go to 22
      if(ix.gt.1) go to 23
      perf=pf
      go to 20
23    itemp=ix/2-(ix-1)/2
      perf=perf+pf*(2.*itemp+2.)
      go to 20
22    perf=(perf+pf)*delt/3.
20    continue
     xa(1)=0.
     xa(2)=0.
     xa(3)=0.
     xa(4)=0.
     xa(5)=0.
     xa(6)=0.
     xa(7)=0.
     xa(8)=0.
     xa(9)=0.
     xa(10)=0.
     xa(11)=0.
     xa(12)=0.
     xa(13)=0.
     xa(14)=0.
     xb(1)=0.
     xb(2)=0.
     xb(3)=0.
     xb(4)=0.
     dm=-delt
      do 30 i=1,ifn
      t=float(ifn-i)*delt
      do 27 k=1,n
      sx(k)=xr(k,ifn+1-i)
 27   continue
      do 47 km=1,m
      sy(km)=yr(km,ifn+1-i)
 47   continue
      if(i.eq.1) go to 31
```

```
      dm=-delt
      call runku(xa,xb,p,n,m,dm,2,sx,sy,t,u,ps,xl)
31    call gradnt(xa,xb,sx,sy,p,u,ps,gp,xl)
      if(i.eq.ifn) go to 32
       if(i.gt.1) go to 33
       do 41 m1=1,np
       grad(m1)=gp(m1)
41     continue
       go to 30
33      itemp=i/2-(i-1)/2
       do 42 m2=1,np
        grad(m2)=grad(m2)+gp(m2)*(2.*itemp+2.)
42     continue
       go to 30
32      do 43 m3=1,np
         grad(m3)=(grad(m3)+gp(m3))*delt/3.
43      continue
30      continue
       do 44 m4=1,np
       grad(m4)=grad(m4)*w(m4)
44     continue
       return
       end
c SUBROUTINE FOR STATE EQUATIONS
      subroutine confun(x,dx,y,p,u,ps,xl)
      dimension x(14),dx(14),y(4),p(4)
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      common /p2/rho,c1,c2,c3,c4
      rho=8e-5
      xk1=p(3)
      xkp=p(4)
      radius=p(1)
      area=3.141592*radius*radius
      c0=sqrt(b/rho)
      te=xl/c0
      zs=sqrt(rho*b)/area
      ys=1./zs
      xnu=xmu/rho
      pe=xnu*xl/c0/p(1)/p(1)
c --- Curve-fit of omega & zeta of line
      c1=.7138
      c2=.5273
      c3=1.3978
      c4=-.8403
      pe2=c4*pe
      z=c1*pe**c2
      w=c0*c3*exp(pe2)/xl
      w2=w*w
      dx(1)=x(2)
      dx(2)=-2.*w2*x(1)-2.*z*w*x(2)+w2*x(3)+w2*zs*te*y(1)
      dx(3)=x(2)+y(3)
      dx(4)=x(5)
      dx(5)=-2.*w2*x(4)-2.*z*w*x(5)+w2*x(6)-w2*ys*te*y(3)
      dx(6)=x(5)+y(1)
```

```
      dx(7)=x(8)
      tem1=w2*ys*te/xk2-2.*z*w
      tem2=-cd*xmu*p(2)*x(13)+p(2)*(1.-cf)*x(12)
      dx(8)=-w2*x(7)+tem1*x(8)+w2*y(2)-xk*xk1/xk2/xj*tem2-
     +  2.*z*w*xk*xk1/xk2*x(13)-w2*xk1*y(4)
      dx(9)=x(10)
      dx(10)=-2.*w2*x(9)-2.*z*w*x(10)+w2*x(11)+w2*zs*te*y(2)
      dx(11)=x(10)+x(7)
      dx(12)=2.*b/v*(-p(2)*x(13)-cs*p(2)/xmu*x(12)+x(5))
      dx(13)=1./xj*tem2
      dx(14)=x(13)
      return
      end
c SUBROUTINE FOR ALGEBRAIC EQUATIONS
      subroutine confuy(y,dy,x,p,u,ps,delt,xl)
      dimension y(4),dy(4),x(14),p(4)
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      common /p2/rho,c1,c2,c3,c4
      xk1=p(3)
      xkp=p(4)
      radius=p(1)
      area=3.141592*radius*radius
      c0=sqrt(b/rho)
      te=xl/c0
      zs=sqrt(rho*b)/area
      ys=1./zs
      xnu=xmu/rho
      pe=xnu*xl/c0/p(1)/p(1)
      pe2=c4*pe
      z=c1*pe**c2
      w=c0*c3*exp(pe2)/xl
      w2=w*w
      tem1=-2.*w2*x(1)-2.*z*w*x(2)+w2*x(3)+w2*zs*te*y(1)
      tem2=-2.*w2*x(4)-2.*z*w*x(5)+w2*x(6)-w2*ys*te*y(3)
      tem3=-2.*w2*x(9)-2.*z*w*x(10)+w2*x(11)+w2*zs*te*y(2)
      tem4=-p(2)*x(13)-cs*p(2)/xmu*x(12)+x(5)
      dy(1)=-xk*xk1*x(13)+xk2*tem1
      dy(2)=tem2
      dy(3)=tem3+b/v*tem4
      dy(4)=-xk*x(13)-xkp*b/v*tem4*2.
      return
      end
c SUBROUTINE FOR ADJOINT STATE EQUATIONS
      subroutine adfun1(xa,dxa,xb,p,x,y,t,u,ps,xl)
      dimension xa(14),dxa(14),xb(4),p(4),x(14),y(4)
      common /c3/aa,omega,zeta,z
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      common /p2/rho,c1,c2,c3,c4
      call desired(aa,omega,zeta,t,dsy)
      xk1=p(3)
      xkp=p(4)
      radius=p(1)
      area=3.141592*radius*radius
      c0=sqrt(b/rho)
```

```
      te=xl/c0
      zs=sqrt(rho*b)/area
      ys=1./zs
      xnu=xmu/rho
      pe=xnu*xl/c0/p(1)/p(1)
      pe2=c4*pe
      z=c1*pe**c2
      w=c0*c3*exp(pe2)/xl
      w2=w*w
      dxa(1)=2.*w2*xa(2)+2.*xk2*w2*xb(1)
      dxa(2)=-xa(1)+2.*z*w*xa(2)-xa(3)+2.*xk2*z*w*xb(1)
      dxa(3)=-w2*xa(2)-xk2*w2*xb(1)
      dxa(4)=2.*w2*xa(5)+2.*w2*xb(2)
      dxa(5)=-xa(4)+2.*z*w*xa(5)-xa(6)-2.*b/v*xa(12)+2.*z*w
     +  *xb(2)-2.*b/v*xb(3)+xb(4)*xkp*b/v*2.
      dxa(6)=-w2*xa(5)-w2*xb(2)
      dxa(7)=w2*xa(8)-xa(11)
      dxa(8)=-xa(7)+(-w2*ys*te/xk2+2.*z*w)*xa(8)
      dxa(9)=2.*w2*xa(10)+2.*w2*xb(3)
      dxa(10)=-xa(9)+2.*z*w*xa(10)-xa(11)+2.*z*w*xb(3)
      dxa(11)=-w2*xa(10)-w2*xb(3)
      tem1=p(2)*(1.-cf)/xj
      tem2=cs*p(2)/xmu
      tem3=b/v*tem2
      tem4=p(2)*tem1-tem2*tem2*b/v
      dxa(12)=xk*xk1/xk2*tem1*xa(8)+2.*tem3*xa(12)-tem1*xa(13)
     +  +2.*tem3*xb(3)-xb(4)*xkp*b/v*tem2*2.
      dxa(13)=xk*xk1/xk2*(-cd*xmu*p(2)/xj+2.*z*w)*xa(8)+2.*
     +  b*p(2)/v*xa(12)+cd*xmu*p(2)/xj*xa(13)-xa(14)+
     + xk*xk1*xb(1)+2.*b*p(2)/v*xb(3)+xk*xb(4)-
     + xb(4)*xkp*b/v*p(2)*2.
      if(x(14).ge.dsy) then
      dxa(14)=1.
      else
      dxa(14)=-1.
      endif
      return
      end
c SUBROUTINE FOR ADJOINT ALGEBRAIC EQUATIONS
      subroutine adfun2(xb,dxb,xa,p,x,y,t,u,ps,xl)
      dimension xb(4),dxb(4),xa(14),p(4),x(14),y(4)
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      common /p2/rho,c1,c2,c3,c4
      radius=p(1)
      dm=p(2)
      xk1=p(3)
      xkp=p(4)
      area=3.141592*radius*radius
      c0=sqrt(b/rho)
      te=xl/c0
      zs=sqrt(rho*b)/area
      ys=1./zs
      xnu=xmu/rho
      pe=xnu*xl/c0/p(1)/p(1)
```

```
      pe2=c4*pe
      z=c1*pe**c2
      w=c0*c3*exp(pe2)/xl
      w2=w*w
      dxb(1)=-w2*zs*te*xa(2)-xa(6)-xk2*w2*zs*te*xb(1)
      dxb(2)=-w2*xa(8)-w2*zs*te*xa(10)-w2*zs*te*xb(3)
      dxb(3)=-xa(3)+w2*ys*te*xa(5)+w2*ys*te*xb(2)
      dxb(4)=w2*xk1*xa(8)
      return
      end
c SUBROUTINE FOR GRADIENT VECTOR
      subroutine gradnt(xa,xb,x,y,p,u,ps,gp,xl)
      dimension xa(14),xb(4),x(14),y(4),p(4),gp(4)
      common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
      common /p2/rho,c1,c2,c3,c4
      radius=p(1)
      dm=p(2)
      xk1=p(3)
      xkp=p(4)
      area=3.141592*radius*radius
      c0=sqrt(b/rho)
      te=xl/c0
      zs=sqrt(rho*b)/area
      ys=1./zs
      xnu=xmu/rho
      c1=xnu*xl/c0
      pe=c1/p(1)/p(1)
      pe2=c4*pe
      z=c1*pe**c2
      w=c0*c3*exp(pe2)/xl
      w2=w*w
      c2m=c2-1.
      dz=c1*c2*pe**(c2m)*c1*(-2.)/p(1)/p(1)/p(1)
      dw=-2.*c4*c1*w/p(1)/p(1)/p(1)
      dzw=dz*w+z*dw
      dwt=2.*w*dw*te
      w2w=2.*w*dw
      w4w=4.*w*dw
      z2=2.*dzw
      zsdw=zs*dwt
      ysdw=ys*dwt
      tem1=ysdw/xk2-z2
      tema=-w4w*x(1)-z2*x(2)+w2w*x(3)+zsdw*y(1)
      temb=-w4w*x(4)-z2*x(5)+w2w*x(6)-ysdw*y(3)
      temc=-w4w*x(9)-z2*x(10)+w2w*x(11)+zsdw*y(2)
      gp(1)=xa(2)*tema+xa(5)*temb+xa(8)*(-w2w*x(7)+tem1*x(8)+
     + w2w*y(2)-z2*xk*xk1/xk2*x(13)-w2w*xk1*y(4))+xa(10)*temc+
     + xb(1)*xk2*tema+xb(2)*temb+xb(3)*temc
      tem2=cs*b/xmu/v
      tem3=-b/v*x(13)-tem2*x(12)
      tem4=-cd*xmu*p(2)*x(13)+p(2)*(1.-cf)*x(12)
      tem5=2.*tem4/xj
      gp(2)=xa(8)*xk*xk1/xk2/xj*(cd*xmu*x(13)-(1.-cf)*x(12))+2.*
     + xa(12)*tem3+xa(13)*(-cd*xmu/xj*x(13)+(1.-
```

```
     + cf)/xj*x(12))+2.*
     + xb(3)*tem3+2.*xb(4)*xkp*b/v*(x(13)+cs/xmu*x(12))
      gp(3)=xa(8)*(-xk/xk2/xj*tem4-2.*z*w*xk/xk2*x(13)-
     + w2*y(4))-xb(1)*xk*x(13)
      tem6=-p(2)*x(13)-cs*p(2)/xmu*x(12)+x(5)
       gp(4)=-xb(4)*b/v*tem6*2.
      return
      end
c SUBROUTINE FOR BRANCHING
      subroutine branch(p,branb,branc)
      dimension p(4),branb(2),branc(2),fxpb(7),fxpc(9)
      data fxpb/.009549,.01512,.030,.05841,.09549,.1512,.3/
      data fxpc/.075,.1,.125,.15,.175,.2,.225,.25,.275/
      maxb=6
      maxc=8
      if (p(2).le.fxpb(1)) then
        branb(1)=fxpb(I)
        branb(2)=fxpb(1)
        go to 100
      endif
      do 10 i=1,maxb
      if (p(2).ge.fxpb(i).and.p(2).le.fxpb(i+1)) then
        branb(1)=fxpb(i)
        branb(2)=fxpb(i+1)
      go to 100
      endif
  10  continue
        branb(1)=fxpb(maxb+1)
        branb(2)=fxpb(maxb+1)
 100  if (p(3).le.fxpc(1)) then
        branc(1)=fxpc(1)
        branc(2)=fxpc(1)
        go to 200
      endif
      do 20 j=1,maxc
      if (p(3).ge.fxpc(j).and.p(3).le.fxpc(j+1)) then
        branc(1)=fxpc(j)
        branc(2)=fxpc(j+1)
        go to 200
      endif
  20  continue
        branc(1)=fxpc(maxc+1)
        branc(2)=fxpc(maxc+1)
 200  return
      end
c SUBROUTINE FOR CHOOSING BRANCH
       subroutine choose(p,ichoose)
       dimension p(4),fxpb(7),fxpc(9)
       data fxpb/.009549,.01512,.030,.05841,.09549,.1512,.3/
       data fxpc/.075,.1,.125,.15,.175,.2,.225,.25,.275/
       ichoose=1
       maxb=7
       do 10 ix=1,maxb
        if(p(2).eq.fxpb(ix)) then
```

```
        ichoose=2
        endif
  10    continue
        return
        end
c SUBROUTINE FOR CHECKING BRANCHING
        subroutine check(p,perf,ichoose,n,m,np,u,ps)
        dimension
        p(4),branb(2),branc(2),xp(101),yp(101),x(14),y(4)
        common /p1/b,v,xj,xmu,cd,cf,cs,xk,xk2
        character*1 answ2
        if(perf.gt.3.) then
        return
        endif
        call branch(p,branb,branc)
        if(ichoose.eq.1) then
         ptm1=abs(p(3)-branc(1))/branc(1)
         ptm2=abs(p(3)-branc(2))/branc(2)
           if(ptm1.le.0.05) then
           p(3)=branc(1)
           go to 10
           else if(ptm2.le.0.05) then
           p(3)=branc(2)
           go to 10
           endif
           return
        else
         ptm1=abs(p(2)-branb(1))/branb(1)
         ptm2=abs(p(2)-branb(2))/branb(2)
           if(ptm1.le.0.05) then
           p(2)=branb(1)
           go to 10
           else if(ptm2.le.0.05) then
           p(2)=branb(2)
           go to 10
           endif
           return
        endif
  10    write(6,120) p(1),p(2),p(3),p(4)
 120    format(3x,'Optimized parameters are'/
     +         5x,5f12.6)
        write(6,121)
 121    format(3x,'Do you want the plot of the optimized
     + response?')
        read(5,122) answ2
 122    format(a1)
         if(answ2.eq.'y') then
       delt=.0001
       xk1=p(3)
       xkp=p(4)
       x(1)=0.
       x(2)=0.
       x(3)=0.
       x(4)=0.
```

```
      x(5)=0.
      x(6)=0.
      x(7)=0.
      x(8)=0.
      x(9)=0.
      x(10)=0.
      x(11)=0.
      x(12)=0.
      x(13)=0.
      x(14)=0.
      y(4)=u-xk*x(14)-xkp*x(12)
      y(1)=xk1*y(4)+xk2*x(2)
      y(2)=x(5)-cs*p(2)/xmu*x(12)
      y(3)=x(10)+x(12)
          do 20 im=1,101
          t=float(im-1)*delt
          call runku(x,y,p,n,m,delt,1,x,y,t,u,ps)
          xp(im)=t
          yp(im)=x(1)
20        continue
       call qckplt(xp,yp,101,'time$','motor
     +  speed$','Response$',0,5,0)
       stop
       endif
       return
       end
c SUBROUTINE FOR DESIRED RESPONSE
       subroutine desired(aa,omega,zeta,t,z)
       r1=sqrt(1./zeta/zeta-1.)
       phi=atan(r1)
       omd=omega*sqrt(1.-zeta*zeta)
       z1=-zeta*omega*t
       z2=exp(z1)
       z3=z2/sin(phi)
       z4=omd*t+phi
       z5=sin(z4)
       z=aa*(1.-z3*z5)
       return
       end
```

VITA

Taijoon Um

Candidate for the Degree of

Doctor of Philosophy

Thesis: OPTIMIZATION OF DESIGN PARAMETERS OF COMPLEX HYDRAULIC SYSTEMS

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Seoul, Korea, April 9, 1954, the son of Sejin and Yoobok Um. Married to Hyesun Lee on May 30, 1981

Education: Graduated from Seoul High School, Seoul, Korea, in February, 1973; received Bachelor of Science degree in Mechanical Engineering from Seoul National University in February, 1977; received Master of Science degree from Korea Advanced Institute of Science and Technology in February, 1979; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in May, 1989.

Professional Experience: Research Engineer, Daewoo Heavy Industries, March, 1979, to July, 1982; Teaching Assistant, Department of Mechanical Engineering, Oklahoma State University, May, 1983, to December, 1988.